COORDINATED HIGHWAYS ACTION RESPONSE TEAM
STATE HIGHWAY ADMINISTRATION

R1B2A Detailed Design

Contract DBM-9713-NMS
TSR # 9803444
Document # M303-DS-005R0

October 26, 2000
By
Computer Sciences Corporation
PB Farradyne Inc.
Integrated Technology Solutions Inc.

Revision	Description	Pages Affected	Date
0	Initial Release	All	October 26, 2000

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document describes the detailed design of the CHART II system software for Release 1, Build 2A. This design refines the high level design presented in document M303-DS-004, *"R1B2A High Level Design,"* to show details regarding the implementation of the high level design. This software release adds functionality to the CHART II system to allow RTMS devices to be configured via the CHART II GUI and to allow data collected from the RTMS devices to be passed to the CHART web site.

1.2 Objectives

The main objective of this design is to provide software developers with details regarding the implementation of the system components described in the high level design to fit within the existing CHART II R1B2 system.

1.3 Scope

This design is limited to components needed to fulfill the requirements of release 1, build 2A of the CHART II system. This design is an add-on to the existing CHART II R1B2 system and therefore detailed design contained in the "R1B2 Servers Detailed Design" and "R1B2 GUI Detailed Design" documents should be referenced for background information pertaining to this design.

This design document does not include design of the database schema used to persist data.

1.4 Design Process

As in the high level design, object-oriented analysis and design techniques were used in creating this design. As such, much of the design is documented using diagrams that conform to the Unified Modeling Language (UML), a de facto standard for diagramming object-oriented designs.

In the high level design, system interfaces were identified and specified in a package named TSSManagement. The interfaces from the high level design were brought forward and enhanced slightly with more detail. The TSSManagementModule package was created to provide the implementation of these interfaces. The GUITSSModule package was created to provide a GUI module to allow Transportation Sensor Systems to be viewed and configured via the CHART II GUI.

Each of these packages (TSSManagementModule and GUITSSModule) are addressed separately in this design with their own class diagram(s) and sequence diagrams for major operations included in the package's interfaces.

The design process for each package involved starting with a class diagram including interfaces from the high level design, and filling in details to the class diagram to move toward

implementation. Sequence diagrams were then used to show how the functionality is to be carried out. An iterative process was used to enhance the class diagram as sequence diagrams identified missing classes or methods.

1.5 Design Tools

The work products contained within this design are extracted from the COOL:JEX design tool. Within this tool, the design is contained in the CHART II project, R1B2 configuration, System Design phase. A system version is included for each software package.

1.6 Work Products

This design contains the following work products:

- A UML Class diagram for each package showing the low-level software objects that will allow the system to implement the interfaces identified in the high level design.
- UML Sequence diagrams for non-trivial operations of each interface identified in the high level design. Additionally, sequence diagrams are included for non-trivial methods in classes created to implement the interfaces. Operations that are considered trivial are operations that do nothing more than return a value or a list of values and where interaction between several classes is not involved.

2 Key Design Concepts

This design builds upon the "R1B2 Servers Detailed Design" and "R1B2 GUI Detailed Design" documents. These documents should be referenced for details on the CHART II Server and GUI frameworks and supporting packages. This section relates key design concepts included in the R1B2 Detailed design documents to their usage in R1B2A.

2.1 Service Application Framework

The service application framework that exists in CHART II (Chart2Service) is used to host the TSSManagementModule, providing access to common services needed by this module. This includes service application maintenance features built into the Chart2Service that allow an administrator to monitor services.

2.2 GUI Application Framework

The existing CHART II core GUI framework is used to host the GUITSSModule, a standard CHART II installable GUI module. This module provides GUI support to allow users to view and configure Transportation Sensor System devices (such as RTMS).

2.3 Access Control

The existing access control features of CHART II R1B2 apply to R1B2A. Any action that may be performed by the user to change the state of the system requires a functional right to be assigned to the user. The GUI disables menus that perform functions for which the user does not have the proper functional rights. The server also blocks attempts by users to execute methods for which they do not have rights, by throwing an AccessDenied exception. A list of specific functional rights added for R1B2A is given in Appendix A of this document.

2.4 Operations Logging

R1B2A uses the existing CHART II operations logging routines to record actions performed by users. Specifically, an entry will be made in the operations log when a user adds an RTMS, removes an RTMS, changes the operations mode of an RTMS (online, offline, maintenance mode), or changes configuration values for an RTMS. When configuration values are changed, the old and new values are logged.

2.5 Event Channel Fault Tolerance

The TSSManagementModule and GUITSSModule both utilize existing utilities in the CHART II system to maintain their connections to the CORBA event service. This ensures that even if a CORBA event service application is stopped, asynchronous events will be enabled automatically within the server and GUI when the event service is restarted.

Note, the server requires the CORBA event service to be running when it is started, however the GUI does not have a start-up dependency on the event service.

2.6 Object Publication

The TSSManagementModule (like other CHART II server modules) publishes objects in the CORBA trader to make them available to other applications, such as the CHART II GUI or the Chartweb.TSSClient. The following object types are published in addition to the existing published CHART II objects:

- TransportationSensorSystem
- RTMS (subclass of TransportationSensorSystem)
- TransportationSensorSystemFactory
- RTMSFactory (subclass of TransportationSensorSystemFactory)

2.7 Database Access

The R1B2A TSSManagementModule uses the existing CHART II DBConnectionManager class to access the CHART II database to retrieve previously persisted TSS objects and to persist TSS objects and their current state. All database access is encapsulated in the TSSManagementDB class.

2.8 Field Communications

The R1B2A TSSManagementModule requires field communications to access RTMS devices and obtain the current traffic parameters they have collected. The distributed CommService and Port objects included in FMS R1B2 provide the field communications for R1B2A. A utility class named RTMSProtocolHandler exists in R1B2A to handle the protocol required to communicate with an RTMS device.

2.9 Error Processing

Because CHART II is a distributed object system, it is expected that any call to a remote object could cause a CORBA exception to be thrown. All software calls to remote objects handle CORBA exceptions and the processing is not shown on sequence diagrams within this design except where it serves to illustrate a design point.

Furthermore, as with any system, most method calls, system calls, etc. can fail unexpectedly. All such errors are handled by the software and are not shown explicitly in the package design portion of this document. The default action when such an error is encountered is to reach a consistent state within the object where the error occurred and then to throw a CHART2Exception (even for non-CORBA calls). The CHART2Exception contains debugging information as well as text suitable for display to a user or administrator. These exceptions are shown on sequence diagrams to call out error conditions that are not obvious.

The Log utility class is used by modules to log error conditions to a flat file that is created by the service application hosting the module. The log file entries contain the name of the class that logged the entry, the date and time of the entry, and descriptive text of the error that occurred. The Log utility also provides the capability for a stack trace to be printed to the file to

accompany the error. This feature is reserved for use when an error condition is caught and the exact cause of the error condition is not known. Log files created by the Log utility class are self-cleaning and are automatically removed from the system when they reach a certain age, as specified in a configuration file.

2.10 Packaging

This software design is broken into packages of related classes. The table below shows each of the packages to be added to CHART II for R1B2A along with a description of each.

Table 1 Package Descriptions

Package Name	Package Description
TSSManagement	This package contains code that is generated from IDL. It contains the CORBA interfaces, structs, enums, and constants used to define the interface between the CHART II TSS service and other applications such as the CHART II GUI and the Chartweb.TSSClient.
TSSManagementModule	This package contains a service application module that implements the interfaces defined in the TSSManagement package.
GUITSSModule	This package contains classes that allow TSS devices served by the TSSManagementModule to be viewed and configured in the CHART II GUI .
DeviceUtility	This package exists in CHART II R1B2, however portions of it relating to the PortLocator are shown in this document to provide a more complete view of device communications.
CHARTWebModule	This package contains classes implementing the CORBA PushConsumer interface and is therefore a CORBA object that is connected to the ORB and is called remotely by an EventChannel (via its PushConsumer push() method) when data is pushed on the channel. This module connects to the TSS status and event channels that exist in the system to allow this module to be notified of status and configuration changes to TSS objects.
CHARTWebMapServerModule	This package contains classes implementing the MapServer that will render the CHART Web Map with the speed information from the RTMS units

The remainder of this document contains detailed designs of each of the above packages.

3 Package Designs

The following sections provide detailed designs of each of the software packages added to CHART II for R1B2A. Each section contains one or more class diagrams. Sequence diagrams exist for non-trivial operations for a software package, except for the TSSManagement package that serves to define the CORBA interfaces to the system.

3.1 TSSManagement

This section shows interfaces to the system that are defined in IDL.

3.1.1 TSSManagement (Class Diagram)

This class diagram contains the interfaces, structs, and typedefs that are to be defined in IDL and provide the external interface to the TSSManagement package of the CHART II system.

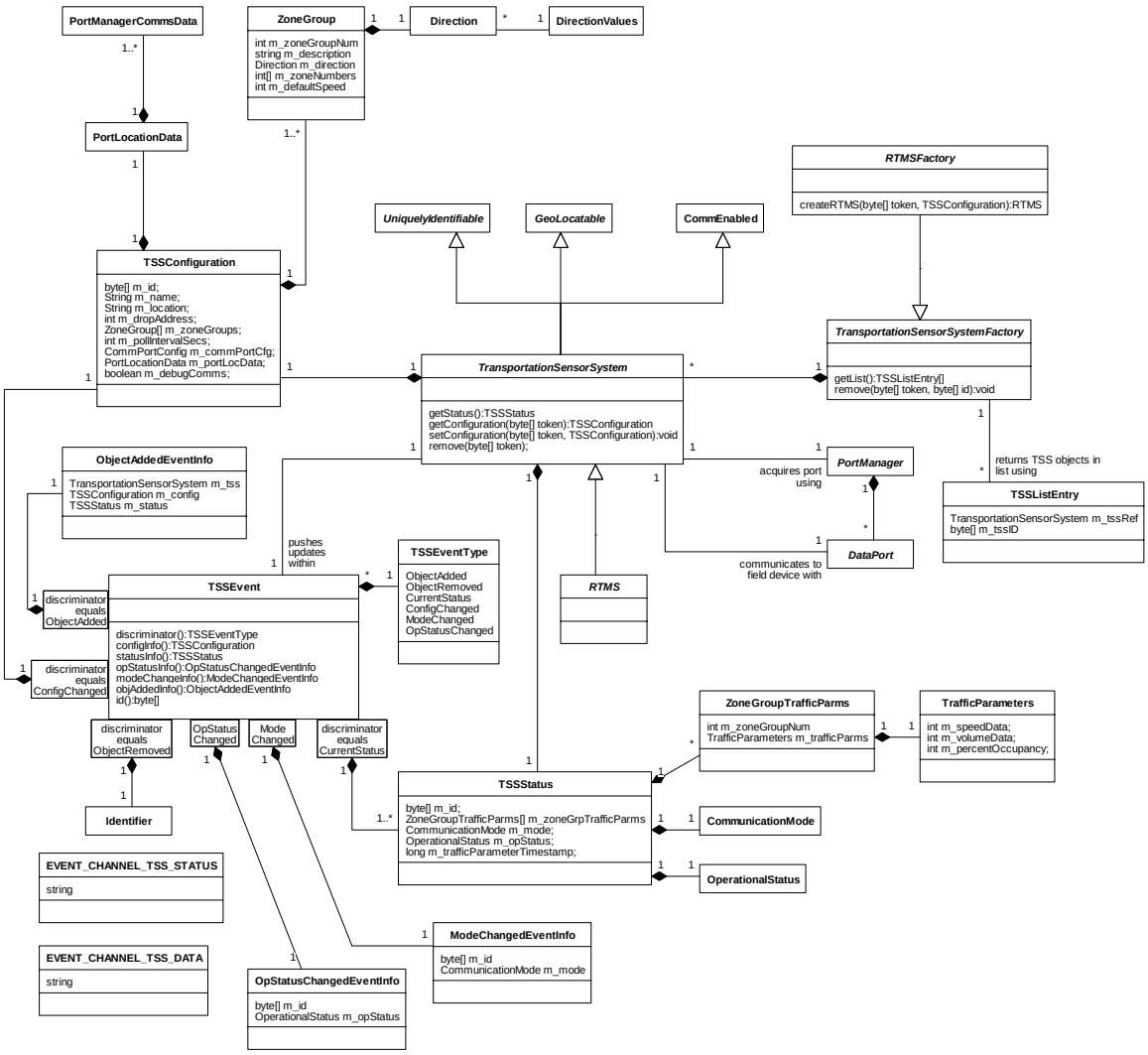

Figure 1. TSSManagement (Class Diagram)

3.1.1.1 CommEnabled (Class)

The CommEnabled interface is implemented by objects that can be taken offline, put online, or put in maintenance mode. These states typically apply only to field devices. When a device is taken offline, it is no longer available for use through the system and automated polling is halted. When put online, a device is again available for use through the system and automated polling is enabled (if applicable). When put in maintenance mode a device is offline except that maintenance commands to the device are allowed to help in troubleshooting.

3.1.1.2 CommunicationMode (Class)

The CommunicationMode class enumerates the modes of operation for a device: ONLINE, OFFLINE, and MAINT_MODE. ONLINE is used to indicate the device is available to the operational system. OFFLINE is used to indicate the device is not available to the online system and communications to the device have been disabled. MAINT_MODE is used to indicate that the device is available only for maintenance / repair activities and testing.

3.1.1.3 DataPort (Class)

A DataPort is a port that allows binary data to be sent and received. Ports of this type support a receive method that allows a chunk of all available data to be received. This method prevents callers from having to issue many receive calls to parse a device response. Instead, this receive call returns all available data received within the timeout parameters. The caller can then parse the data within a local buffer. Using this mechanism, device command and response should require only one call to send and one call to receive.

3.1.1.4 Direction (Class)

This type defines a short value that is used to indicate a direction of travel as defined in DirectionValues.

3.1.1.5 DirectionValues (Class)

This interface contains constants for directions as defined in the TMDD.

3.1.1.6 EVENT_CHANNEL_TSS_DATA (Class)

This is a static string that contains the name of the event channel used to push events that contain Transportation Sensor System traffic parameter data. The following TSSEventTypes are pushed on EVENT_CHANNEL_TSS_DATA channels:

- CurrentStatus

3.1.1.7 EVENT_CHANNEL_TSS_STATUS (Class)

This is a static string that contains the name of the event channel used to push events relating to the change in a Transportation Sensor System status and/or configuration. The following TSSEventTypes are pushed on EVENT_CHANNEL_TSS_STATUS channels:

- ObjectAdded
- ObjectRemoved
- ConfigChanged
- ModeChanged
- OpStatusChanged

3.1.1.8 GeoLocatable (Class)

This interface is implemented by objects that can provide location information to their users.

3.1.1.9 Identifier (Class)

Wrapper class for a CHART2 identifier byte sequence. This class will be used to add identifiable objects to hash tables and perform subsequent lookup operations.

3.1.1.10 ModeChangedEventInfo (Class)

This struct contains information pushed with a ModeChanged event.

m_id — The ID of the TSS whose communication mode has changed.

m_mode — The new communication mode for the TSS.

3.1.1.11 ObjectAddedEventInfo (Class)

This structure contains information passed in the ObjectAdded event pushed on a TSS status event channel. It contains the object reference that has been added along with its configuration values and current status values.

3.1.1.12 OperationalStatus (Class)

The OperationalStatus class enumerates the types of operational status a device can have: OK (normal mode), COMM_FAILURE (no communications to the device), or HARDWARE_FAILURE (device is reachable but is reporting a hardware failure).

3.1.1.13 OpStatusChangedEventInfo (Class)

This struct contains data passed with an OpStatusChanged event.

m_id — The ID of the TSS whose operational status has changed.

m_opStatus — The new operational status for the device.

3.1.1.14 PortLocationData (Class)

This class contains configuration data that specifies the communication server(s) to use to communicate with a device.

m_commsData — One or more objects identifying the communications server (PortManager) to use to communicate with the device, in order of preference.

m_portType — The type of port to use to communicate with the device (ISDN modem, POTS modem, direct, etc.)

m_portWaitTimeSecs — The maximum number of seconds to wait when attempting to acquire a port from a port manager.

3.1.1.15 PortManager (Class)

A PortManager is an object that manages shared access to communications port resources. The getPort method is used to request the use of a port from the PortManager. Requests for ports specify the type of port needed, the priority of the request, and the maximum time the requester is willing to wait if a port is not immediately available. When the port manager returns a port, the requester has exclusive use of the port until the requester releases the port back to the PortManager or the PortManager reclaims the port due to inactivity.

3.1.1.16 PortManagerCommsData (Class)

This class contains values that identify a port manager and the phone number to dial to access a device from the given port manager. This class exists to allow for the phone number used to access a device to differ based on the port manager to take into account the physical location of the port manager within the telephone network. For example, when dialing a device from one location the call may be long distance but when dialing from another location the call may be local.

3.1.1.17 RTMS (Class)

The Remote Traffic Microwave Sensor (RTMS) is a detector manufactured by EIS, Inc. capable of providing lane level volume, speed, and occupancy data for up to 8 lanes of a roadway at a single location. This interface serves to identify TransportationSensorSystem objects as being of the type RTMS. It also provides a placeholder for future operations that may not apply to TSS objects in general and are instead RTMS specific.

3.1.1.18 RTMSFactory (Class)

Objects that implement RTMSFactory are capable of adding an RTMS to the system.

3.1.1.19 TransportationSensorSystem (Class)

A Transportation Sensor System (TSS) is a generic term used to describe a class of technology used for detection within the transportation industry. Examples of TSS devices range from the advanced devices, such as RTMS, to basic devices, such as single loop detectors.

This software interface is implemented by objects that provide access to the traffic parameters sensed by a Transportation Sensor System. Transportation Sensor Systems are capable of providing detection for one or more detection zones. A single loop detector would have one detection zone, while an RTMS would have eight detection zones.

3.1.1.20 TransportationSensorSystemFactory (Class)

This interface is implemented by objects that are used to create and serve TransportationSensorSystem (TSS) Objects. All factories of TSS objects can return the list of TSS objects which they have created and serve. Derived interfaces are used to provide factories to create specific make, models, and types of TransportationSensorSystem objects.

3.1.1.21 TrafficParameters (Class)

This struct contains traffic parameters that are sensed and reported by a Traffic Sensor System such as the RTMS.

m_speedData — The arithmetic mean of the speeds collected over a sample period in miles per hour in tenths (thus 550 = 55.0 MPH). Valid values are 0 to 2550. A value of 65535 is used to indicate a missing or invalid value (such as when the volume for the sample period is zero).

m_volumeData — The count of vehicles for the sample period. Valid values 0 to 65535. A value of 65535 represents a missing value.

m_percentOccupancy — The percentage of occupancy of the roadway in tenths of a percent. (thus 1000 = 100.0 percent). Valid values are 0 to 1000. A value of 65535 represents a missing or invalid value.

3.1.1.22 TSSConfiguration (Class)

This class holds configuration data for a transportation sensor system (TSS) as follows:

m_id — The unique identifier for this TSS. This field is ignored when the object is passed to the TSS to change its configuration.

m_name — The name used to identify the TSS.

m_location — A descriptive location of the TSS.

m_dropAddress — The drop address for the device.

m_zoneGroups — Logical groupings of detection zones, used to provide a single set of traffic parameters for one or more detection zones.

m_pollIntervalSecs — The interval on which the TSS should be polled for its current traffic parameters (in seconds).

m_commPortCfg — Communication configuration values.

m_portLocData — Configuration information that determines which port manager(s) should be used to establish a connection with the SensorSystem.

m_debugComms — Flag used to enable/disable the logging of communications data for this TSS. When enabled, command and response packets exchanged with the device are logged to a debugging log file.

3.1.1.23 TSSEvent (Class)

This class is a CORBA union that contains varying data depending on the current value of the discriminator.

If the discriminator is ConfigChanged, this union contains a TSSConfig object.

If the discriminator is ObjectAdded, this union contains an ObjectAddedEventInfo object.

If the discriminator is ObjectRemoved, this union contains a byte[] containing the unique identifier for the Traffic Sensor System that was removed.

If the discriminator is CurrentStatus the union contains an array of one or more TSSStatus objects.

If the discriminator is ModeChanged, the union contains a ModeChangedEventInfo.

If the discriminator is OpStatusChanged, the union contains an OpStatusChangedEventInfo object.

3.1.1.24 TSSEventType (Class)

This enumeration defines the types of events that may be pushed on an event channel by a Transportation Sensor Status object. The values in this enumeration are used as the discriminator in the TSSEvent union.

ObjectAdded — a TransportationSensorSystem has been added to the system.

ObjectRemoved — a TransportationSensorSystem has been removed from the system.

CurrentStatus — The event contains the current status of one or more Transportation Sensor System objects.

ConfigChanged — One or more configuration values for the Transportation Sensor System have been changed.

ModeChanged — The communications mode of the TransportationSensorSystem has changed.

OpStatusChanged — The operational status of the TransportationSensorSystem has changed.

3.1.1.25 TSSListEntry (Class)

This struct is used to pass a TransporationSensorSystem object together with its ID. This struct is provided for convenience because when discovering an object, it is usually required to make a call to the object's getID() method.

3.1.1.26 TSSStatus (Class)

This class holds current status information for the TSS as follows:

m_id — The ID of the TSS for which this status applies.

m_zoneGrpTrafficParms — The traffic parameters for each ZoneGroup of the Transporation Sensor System as specified in the Sensor system's TSSConfiguration object.

m_mode — The communication mode of the TSS.

m_opStatus — The operational status for the TSS.

m_trafficParameterTimestamp — A timestamp that records when the traffic parameter data was collected from the device.

3.1.1.27 UniquelyIdentifiable (Class)

This interface will be implemented by all classes that are to be identifiable within the system. The identifier must be generated by the IdentifierGenerator to ensure uniqueness.

3.1.1.28 ZoneGroup (Class)

This class is used to group one or more detection zones of a Transportation Sensor System into a logical grouping. Traffic parameters for all detection zones included in the group are averaged to provide a single set of traffic parameters for the group.

3.1.1.29 ZoneGroupTrafficParms (Class)

This struct contains traffic parameters for a ZoneGroup.

m_zoneGroupNumber — The number of the zone group for which the traffic parameters apply.

m_trafficParms — The traffic parameter values for the zone group.

3.2 TSSManagementModule

3.2.1 TSSModuleClassDiagram (Class Diagram)

This class diagram shows classes in the TSSManagementModule used to allow the module to run within the CHART II service framework and also to provide common services to other classes within the module.

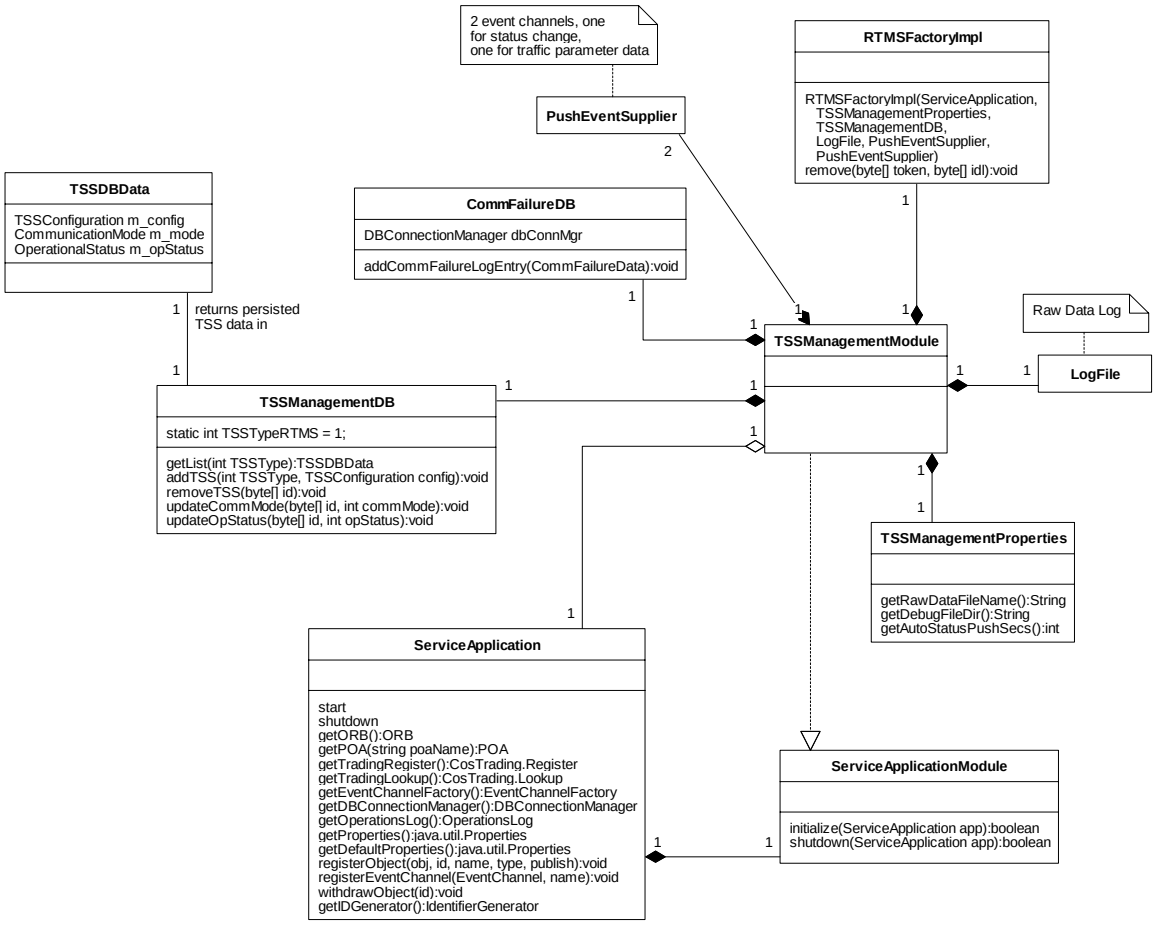

Figure 2. TSSModuleClassDiagram (Class Diagram)

3.2.1.1 CommFailureDB (Class)

This class is a utility used to log an entry in the Comm Failure log table in the database. This table is used to log details about any comm failure that occurs in the system.

3.2.1.2 LogFile (Class)

This class creates a flat file for writing system trace log messages and purges them at user specified interval. The log files created by this class are used for system debugging and maintenance only and are not to be confused with the system operations log that is modeled by the OperationsLog class.

3.2.1.3 PushEventSupplier (Class)

This class provides a utility for application modules that push events on an event channel. The user of this class can pass a reference to the event channel factory to this object. The constructor will create a channel in the factory. The push method is used to push data on the event channel. The push method is able to detect if the event channel or its associated objects have crashed. When this occurs, a flag is set, causing the push method to attempt to reconnect the next time push is called. To avoid a supplier with a heavy supply load from causing reconnect attempts to occur too frequently, a maximum reconnect interval is used. This interval specifies the quickest reconnect interval that can be used. The push method uses this interval and the current time to determine if a reconnect should be attempted, thus reconnects can be throttled independently of a supplier's push rate.

3.2.1.4 RTMSFactoryImpl (Class)

This class implements the RTMSFactory interface as defined in the IDL. It holds all RTMSImpl objects that have been created within an instance of the RTMSManagementModule and allows for the addition and removal of RTMS objects. It also allows one to query all RTMS objects currently served from the factory.

This factory contains a timer that periodically fires, causing the RTMSFactoryImpl to collect the current status of each RTMSImpl and push the collective status in a single CORBA event.

3.2.1.5 ServiceApplication (Class)

This interface is implemented by objects that can provide the basic services needed by a ChartII service application. These services include providing access to basic CORBA objects that are needed by service applications, such as the ORB, POA, Trader, and Event Service.

3.2.1.6 ServiceApplicationModule (Class)

This interface is implemented by modules that serve CORBA objects. Implementing classes are notified when their host service is initialized and when it is shutdown. The implementing class can use these notifications along with the services provided by the invoking ServiceApplication to perform actions such as object creation and publication.

3.2.1.7 TSSDBData (Class)

This class holds data that is retrieved from the database during start-up for a Transportation Sensor System object that existed in the system during a prior run of the software.

3.2.1.8 TSSManagementDB (Class)

This class is a utility that provides methods for adding, removing, and updating database data pertaining to Transportation Sensor Systems. Because this class is designed to be generic and work for RTMS as well as other TSS derived objects, the add method requires a model id to be passed. This allows data for a specific model to be retrieved by model specific factories during system initialization.

3.2.1.9 TSSManagementModule (Class)

This class is a ServiceApplicationModule used to serve an RTMSFactory object. The RTMSFactory serves zero or more RTMS objects. By providing an implementation of the ServiceApplicationModule interface, this class can be included in the CHART2 service application framework, which provides common services needed to serve CORBA objects within the CHART 2 system.

3.2.1.10 TSSManagementProperties (Class)

This class provides a wrapper to the application's properties file that provides easy access to the properties specific to the TSSManagementModule. These properties include the name of the file where raw traffic parameter data is to be logged, the directory where debug log files are to be kept, and the interval at which the status of all TSS objects is to be collected and pushed in a CORBA event.

3.2.2 RTMSFactoryClassDiagram (Class Diagram)

This diagram shows the classes of the TSSManagementModule relating to the RTMSFactoryImpl. The RTMSFactoryImpl holds RTMSImpl objects and allows RTMSs to be added and removed from the system.

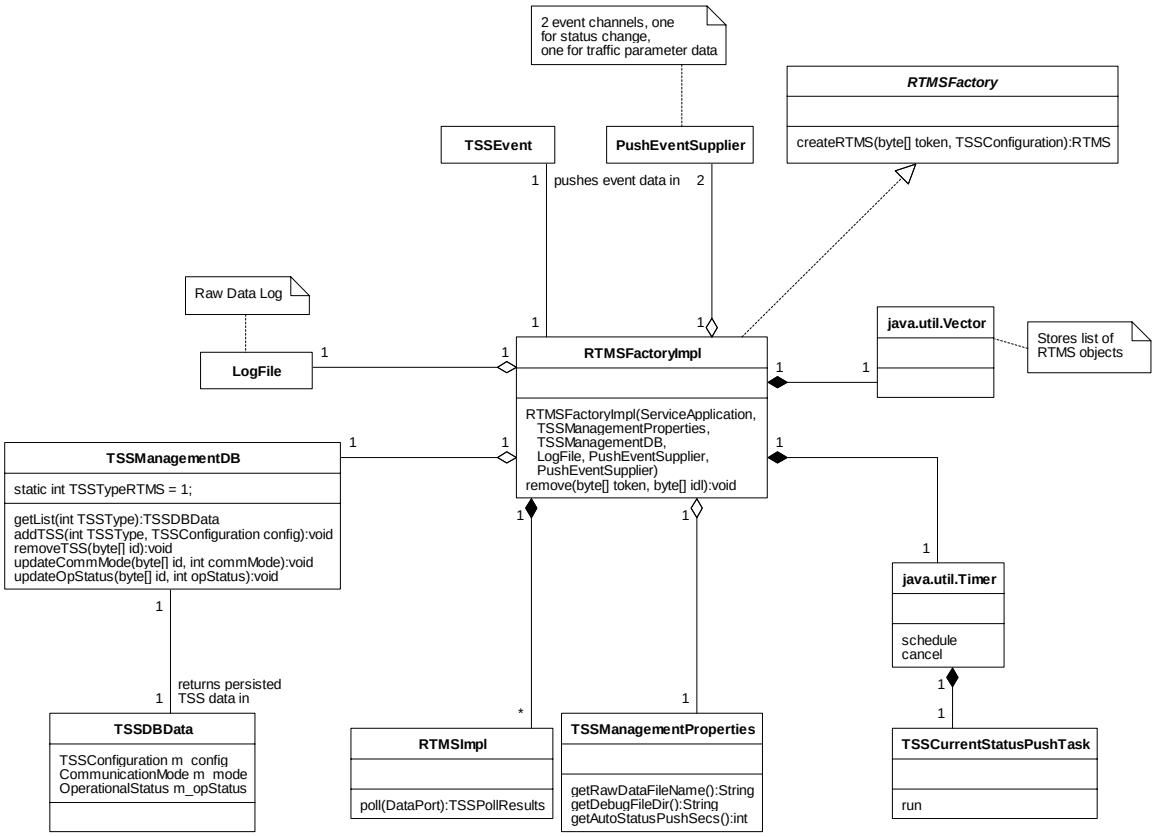

Figure 3. RTMSFactoryClassDiagram (Class Diagram)

3.2.2.1 java.util.Timer (Class)

This class provides asynchronous execution of tasks that are scheduled for one-time or recurring execution.

3.2.2.2 java.util.Vector (Class)

A Vector is a growable array of objects.

3.2.2.3 LogFile (Class)

This class creates a flat file for writing system trace log messages and purges them at user specified interval. The log files created by this class are used for system debugging and maintenance only and are not to be confused with the system operations log that is modeled by the OperationsLog class.

3.2.2.4 PushEventSupplier (Class)

This class provides a utility for application modules that push events on an event channel. The user of this class can pass a reference to the event channel factory to this object. The constructor will create a channel in the factory. The push method is used to push data on the event channel. The push method is able to detect if the event channel or its associated objects have crashed. When this occurs, a flag is set, causing the push method to attempt to reconnect the next time push is called. To avoid a supplier with a heavy supply load from causing reconnect attempts to occur too frequently, a maximum reconnect interval is used. This interval specifies the quickest reconnect interval that can be used. The push method uses this interval and the current time to determine if a reconnect should be attempted, thus reconnects can be throttled independently of a supplier's push rate.

3.2.2.5 RTMSFactory (Class)

Objects that implement RTMSFactory are capable of adding an RTMS to the system.

3.2.2.6 RTMSFactoryImpl (Class)

This class implements the RTMSFactory interface as defined in the IDL. It holds all RTMSImpl objects that have been created within an instance of the RTMSManagementModule and allows for the addition and removal of RTMS objects. It also allows one to query all RTMS objects currently served from the factory.

This factory contains a timer that periodically fires, causing the RTMSFactoryImpl to collect the current status of each RTMSImpl and push the collective status in a single CORBA event.

3.2.2.7 RTMSImpl (Class)

This class is a derivation of the PolledTSSImpl that provides functionality for obtaining the current traffic parameters from an RTMS device. It makes use of an RTMSProtocolHandler to perform the device specific protocol to obtain the traffic parameters. It moves the data from the device specific format to the generic TSSPollResults object to allow the PolledTSSImpl to combine/average data based on zone group configuration, perform raw data logging, and other services that are common to Transportation Sensor System objects.

3.2.2.8 TSSCurrentStatusPushTask (Class)

This class is a timer task that is executed on a regular interval. When this task is run, it calls into the RTMSFactoryImpl object to have it collect the status for all RTMSImpl objects and to push a CurrentStatus event with the collected data.

3.2.2.9 TSSDBData (Class)

This class holds data that is retrieved from the database during start-up for a Transportation Sensor System object that existed in the system during a prior run of the software.

3.2.2.10 TSSEvent (Class)

This class is a CORBA union that contains varying data depending on the current value of the discriminator.

If the discriminator is ConfigChanged, this union contains a TSSConfig object.

If the discriminator is ObjectAdded, this union contains an ObjectAddedEventInfo object.

If the discriminator is ObjectRemoved, this union contains a byte[] containing the unique identifier for the Traffic Sensor System that was removed.

If the discriminator is CurrentStatus the union contains an array of one or more TSSStatus objects.

If the discriminator is ModeChanged, the union contains a ModeChangedEventInfo.

If the discriminator is OpStatusChanged, the union contains an OpStatusChangedEventInfo object.

3.2.2.11 TSSManagementDB (Class)

This class is a utility that provides methods for adding, removing, and updating database data pertaining to Transportation Sensor Systems. Because this class is designed to be generic and work for RTMS as well as other TSS derived objects, the add method requires a model id to be passed. This allows data for a specific model to be retrieved by model specific factories during system initialization.

3.2.2.12 TSSManagementProperties (Class)

This class provides a wrapper to the application's properties file that provides easy access to the properties specific to the TSSManagementModule. These properties include the name of the file where raw traffic parameter data is to be logged, the directory where debug log files are to be kept, and the interval at which the status of all TSS objects is to be collected and pushed in a CORBA event.

3.2.3 RTMSObject (Class Diagram)

This diagram shows classes in the TSSManagementModule relating to the RTMSImpl class. The RTMSImpl obtains most of its functionality from its base class, PolledTSSImpl. The RTMSImpl object provides logic that allows the base class to obtain traffic parameters from an RTMS device.

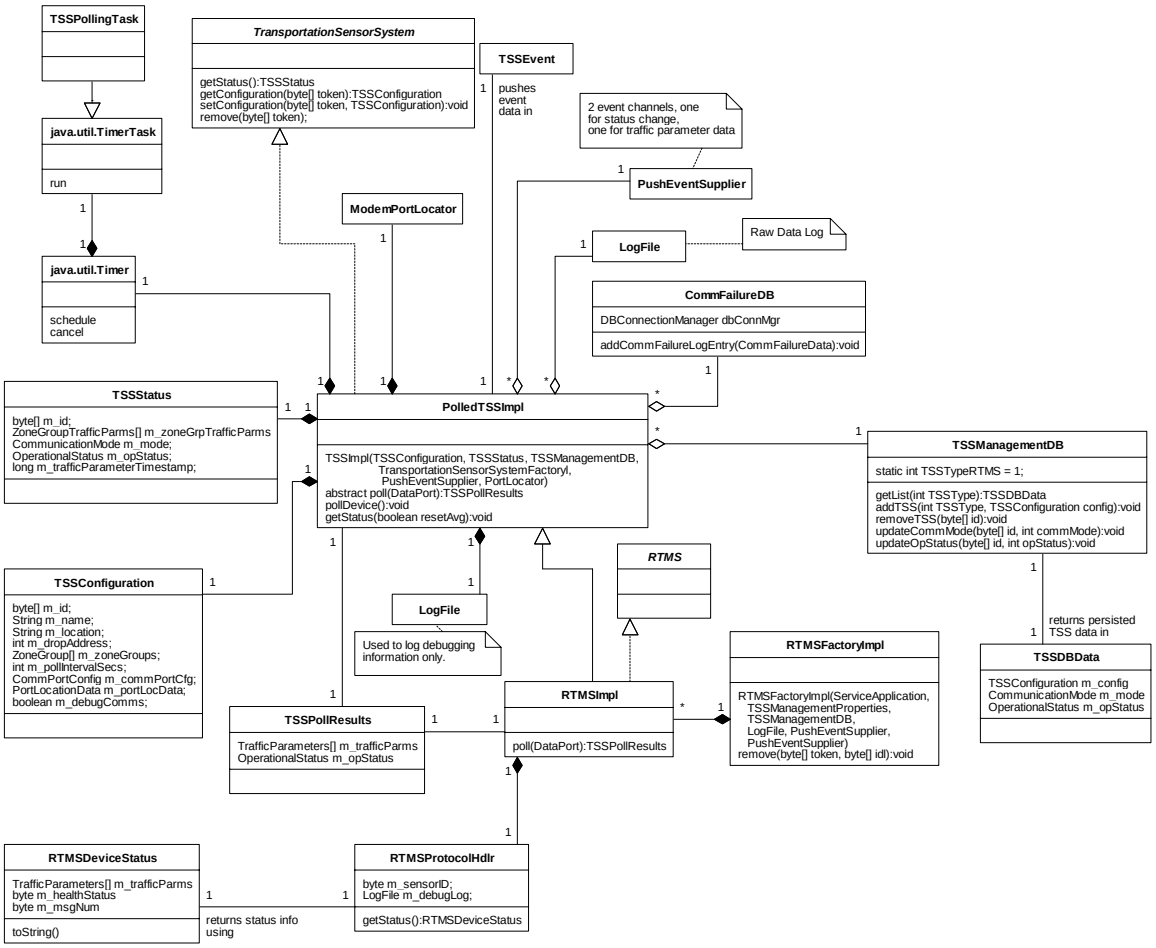

Figure 4. RTMSObject (Class Diagram)

3.2.3.1 CommFailureDB (Class)

This class is a utility used to log an entry in the Comm Failure log table in the database. This table is used to log details about any comm failure that occurs in the system.

3.2.3.2 java.util.Timer (Class)

This class provides asynchronous execution of tasks that are scheduled for one-time or recurring execution.

3.2.3.3 java.util.TimerTask (Class)

This class is an abstract base class which can be scheduled with a timer to be executed one or more times.

3.2.3.4 LogFile (Class)

This class creates a flat file for writing system trace log messages and purges them at user specified interval. The log files created by this class are used for system debugging and maintenance only and are not to be confused with the system operations log that is modeled by the OperationsLog class.

3.2.3.5 ModemPortLocator (Class)

This class provides an implementation of the PortLocator's abstract connectPort() method that can connect a ModemPort that has been acquired by the PortLocator base class. This derived class logs information in the comm failure database table relating to connection problems that may occur.

3.2.3.6 PolledTSSImpl (Class)

This object implements the Transportation Sensor System interface as defined in IDL. This implementation provides the base functionality required for Transporation Sensor Systems that are polled periodically to retrieve traffic parameters. The only requirement for derived classes is to provide an implmentation of the abstract poll method, which communicates over a previously connected Port to obtain the traffic parameters from a TSS.

This implementation periodically polls the field device using the derived class implementation of the poll method. This implementation provides services such as raw data logging, averaging/summation of data into configured zone groups, asynchronous notification of configuration changes, and persistence/depersistence.

3.2.3.7 PushEventSupplier (Class)

This class provides a utility for application modules that push events on an event channel. The user of this class can pass a reference to the event channel factory to this object. The constructor will create a channel in the factory. The push method is used to push data on the event channel. The push method is able to detect if the event channel or its associated objects have crashed. When this occurs, a flag is set, causing the push method to attempt to reconnect the next time push is called. To avoid a supplier with a heavy supply load from causing reconnect attempts to occur too frequently, a maximum reconnect interval is used. This interval specifies the quickest reconnect interval that can be used. The push method uses this interval and the current time to determine if a reconnect should be attempted, thus reconnects can be throttled independently of a supplier's push rate.

3.2.3.8 RTMS (Class)

The Remote Traffic Microwave Sensor (RTMS) is a detector manufactured by EIS, Inc. capable of providing lane level volume, speed, and occupancy data for up to eight lanes of a roadway at a single location. This interface serves to identify TransportationSensorSystem objects as being of the type RTMS. It also provides a placeholder for future operations that may not apply to TSS objects in general and are instead RTMS specific.

3.2.3.9 RTMSDeviceStatus (Class)

This class is used to pass raw data retrieved from the RTMS to the caller of the RTMSProtocolHdlr getStatus() method.

m_trafficParameters — the traffic parameters sensed by the device, such as volume, speed, and occupancy.

m_healthStatus — The health status byte reported from the RTMS. A value other than 10, 20, 30, 40, 50, 60, or 70 indicates a hardware problem.

m_msgNum — The message number reported by the RTMS. This number is incremented sequentially when the RTMS dumps averaged data to a retrieval area at the end of a message period. It can be used to determine if the device is being polled too frequently or infrequently.

3.2.3.10 RTMSFactoryImpl (Class)

This class implements the RTMSFactory interface as defined in the IDL. It holds all RTMSImpl objects that have been created within an instance of the RTMSManagementModule and allows for the addition and removal of RTMS objects. It also allows one to query all RTMS objects currently served from the factory.

This factory contains a timer that periodically fires, causing the RTMSFactoryImpl to collect the current status of each RTMSImpl and push the collective status in a single CORBA event.

3.2.3.11 RTMSImpl (Class)

This class is a derivation of the PolledTSSImpl that provides functionality for obtaining the current traffic parameters from an RTMS device. It makes use of an RTMSProtocolHandler to perform the device specific protocol to obtain the traffic parameters. It moves the data from the device specific format to the generic TSSPollResults object to allow the PolledTSSImpl to combine/average data based on zone group configuration, perform raw data logging, and other services that are common to Transportation Sensor System objects.

3.2.3.12 RTMSProtocolHdlr (Class)

This class is a utility that encapsulates the communication protocol of the RTMS device. It provides a high level method to get the status as an object. It formats a command and sends it to the device and receives and interprets the response from the device, passing the data back to the caller in the form of an RTMSDeviceStatus object.

3.2.3.13 TransportationSensorSystem (Class)

A Transportation Sensor System (TSS) is a generic term used to describe a class of technology used for detection within the transportation industry. Examples of TSS devices range from the advanced devices, such as RTMS, to basic devices, such as single loop detectors.

This software interface is implemented by objects that provide access to the traffic parameters sensed by a Transportation Sensor System. Transportation Sensor Systems are capable of providing detection for one or more detection zones. A single loop detector would have one detection zone, while an RTMS would have 8 detection zones.

3.2.3.14 TSSConfiguration (Class)

This class holds configuration data for a transportation sensor system (TSS) as follows:

m_id — The unique identifier for this TSS. This field is ignored when the object is passed to the TSS to change its configuration.

m_name — The name used to identify the TSS.

m_location — A descriptive location of the TSS.

m_dropAddress — The drop address for the device.

m_zoneGroups — Logical groupings of detection zones, used to provide a single set of traffic parameters for one or more detection zones.

m_pollIntervalSecs — The interval on which the TSS should be polled for its current traffic parameters (in seconds).

m_commPortCfg — Communication configuration values.

m_portLocData — Configuration information that determines which port manager(s) should be used to establish a connection with the SensorSystem.

m_debugComms — Flag used to enable/disable the logging of communications data for this TSS. When enabled, command and response packets exchanged with the device are logged to a debugging log file.

3.2.3.15 TSSDBData (Class)

This class holds data that is retrieved from the database during start-up for a Transportation Sensor System object that existed in the system during a prior run of the software.

3.2.3.16 TSSEvent (Class)

This class is a CORBA union that contains varying data depending on the current value of the discriminator.

If the discriminator is ConfigChanged, this union contains a TSSConfig object.

If the discriminator is ObjectAdded, this union contains an ObjectAddedEventInfo object.

If the discriminator is ObjectRemoved, this union contains a byte[] containing the unique identifier for the Traffic Sensor System that was removed.

If the discriminator is CurrentStatus the union contains an array of one or more TSSStatus objects.

If the discriminator is ModeChanged, the union contains a ModeChangedEventInfo.

If the discriminator is OpStatusChanged, the union contains an OpStatusChangedEventInfo object.

3.2.3.17 TSSManagementDB (Class)

This class is a utility that provides methods for adding, removing, and updating database data pertaining to Transportation Sensor Systems. Because this class is designed to be generic and work for RTMS as well as other TSS derived objects, the add method requires a model id to be passed. This allows data for a specific model to be retrieved by model specific factories during system initialization.

3.2.3.18 TSSPollingTask (Class)

This class is a TimerTask that is used by an RTMS to schedule its asynchronous polling with a Timer object.

3.2.3.19 TSSPollResults (Class)

This class is a data holder used to pass the results of device polling from the PolledTSSImpl derived class back to the base class for processing. The traffic parameter data passed is lane (detection zone) level. The operational status is the status as determined by the derived class.

m_trafficParms — An array of traffic parameters for the current poll cycle, with one array entry for each detection zone of the device.

m_opStatus — The operational status as determined by the derived class.

3.2.3.20 TSSStatus (Class)

This class holds current status information for a TSS as follows:

m_id — The ID of the TSS for which this status applies.

m_zoneGrpTrafficParms — The traffic parameters for each ZoneGroup of the Transporation Sensor System as specified in the Sensor system's TSSConfiguration object.

m_mode — The communication mode of the TSS.

m_opStatus — The operational status for the TSS.

m_trafficParameterTimestamp — A timestamp that records when the traffic parameter data was collected from the device.

3.2.4 Sequence Diagrams

3.2.4.1 PolledTSSImpl:putInMaintenanceMode (Sequence Diagram)

A user with the proper functional rights can put a Transportation Sensor System in maintenance mode if it is not already in maintenance mode. The communication mode stored in the TSSStatus object is updated to indicate maintenance mode. If a polling timer does not already exist, it is created and the TSSPollingTask is scheduled for the configured polling interval. A CORBA event is pushed on the Status event channel to notify others of the change. An entry is made in the operations log to record that the user has performed this action.

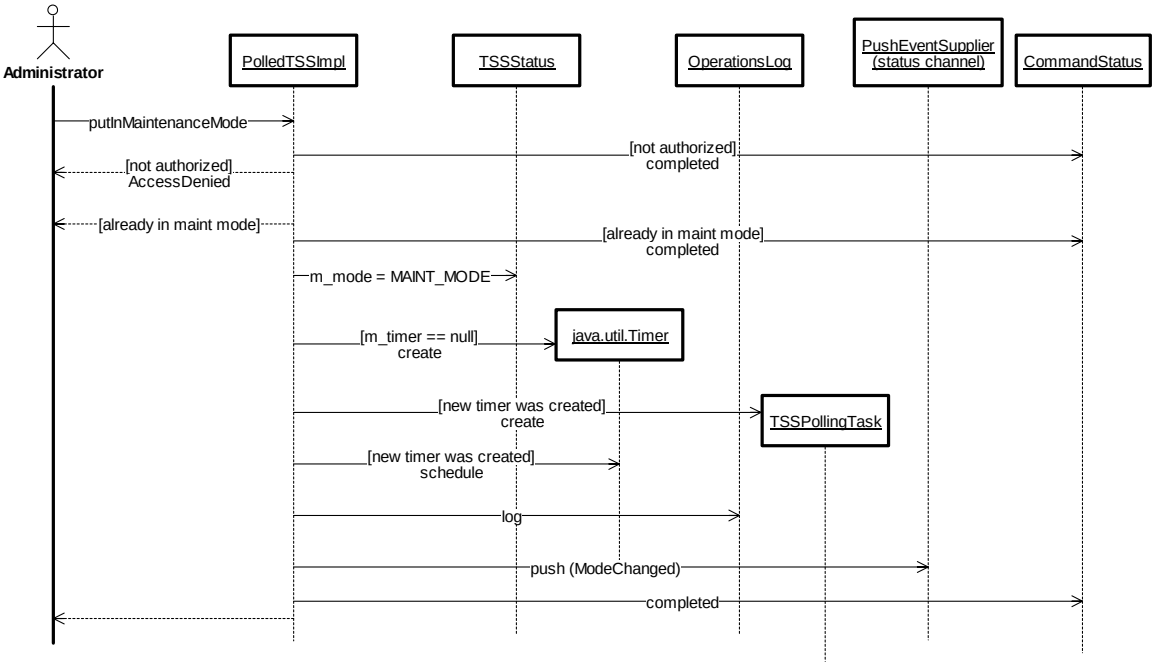

Figure 5. PolledTSSImpl:putInMaintenanceMode (Sequence Diagram)

3.2.4.2 PolledTSSImpl:putOnline (Sequence Diagram)

A user with the proper functional rights can put a Transportation Sensor System online if it is not already online. The communication mode stored in the TSSStatus object is updated to indicate the sensor is online. If a polling timer does not already exist, it is created and the TSSPollingTask is scheduled for the configured polling interval. A CORBA event is pushed on the Status event channel to notify others of the change. An entry is made in the operations log to record that the user has performed this action.

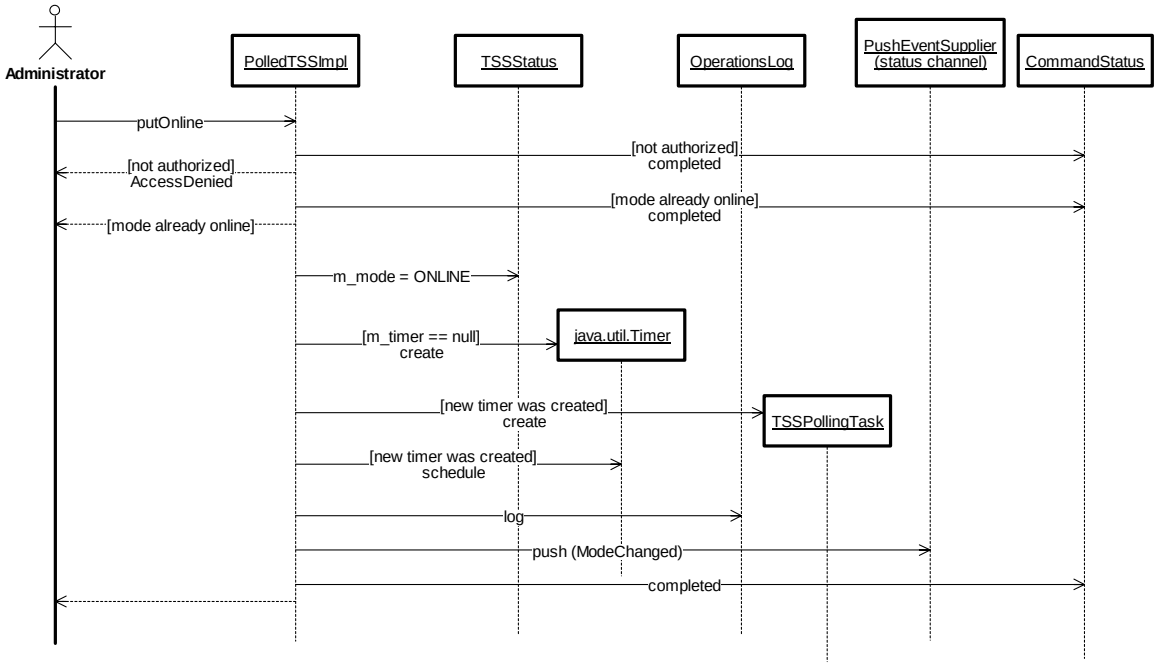

Figure 6. PolledTSSImpl:putOnline (Sequence Diagram)

3.2.4.3 PolledTSSImpl:setConfiguration (Sequence Diagram)

A user with the proper functional rights can change the configuration of a Transportation Sensor System. The previous configuration values are used to detect values that have been changed. If the Port location data has been changed, a new PortLocator object is created with the new values. If the polling interval has been changed and the device is not offline, the existing polling timer is cancelled and destroyed, a new timer is created, and a new polling task is scheduled. If any values were changed, an entry is made in the operations log to record the values that the user has changed. A CORBA event is pushed on the Status event channel to provide notification of the configuration change to other applications.

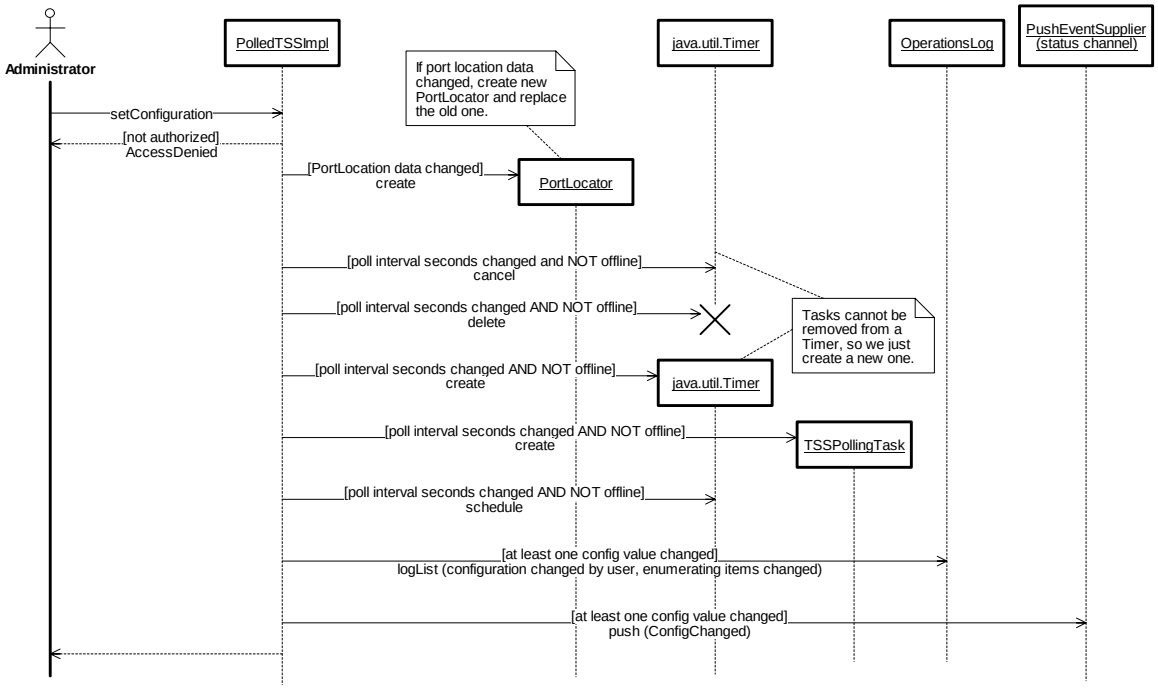

Figure 7. PolledTSSImpl:setConfiguration (Sequence Diagram)

3.2.4.4 PolledTSSImpl:takeOffline (Sequence Diagram)

A user with the proper functional rights can take a Transportation Sensor System offline from the system if it is not already offline. The communication mode stored in the TSSStatus object is updated to indicate the sensor is offline. The timer used to periodically invoke the polling process is cancelled and a CORBA event is pushed on the Status event channel to notify others of the change. An entry is made in the operations log to record that the user has performed this action.

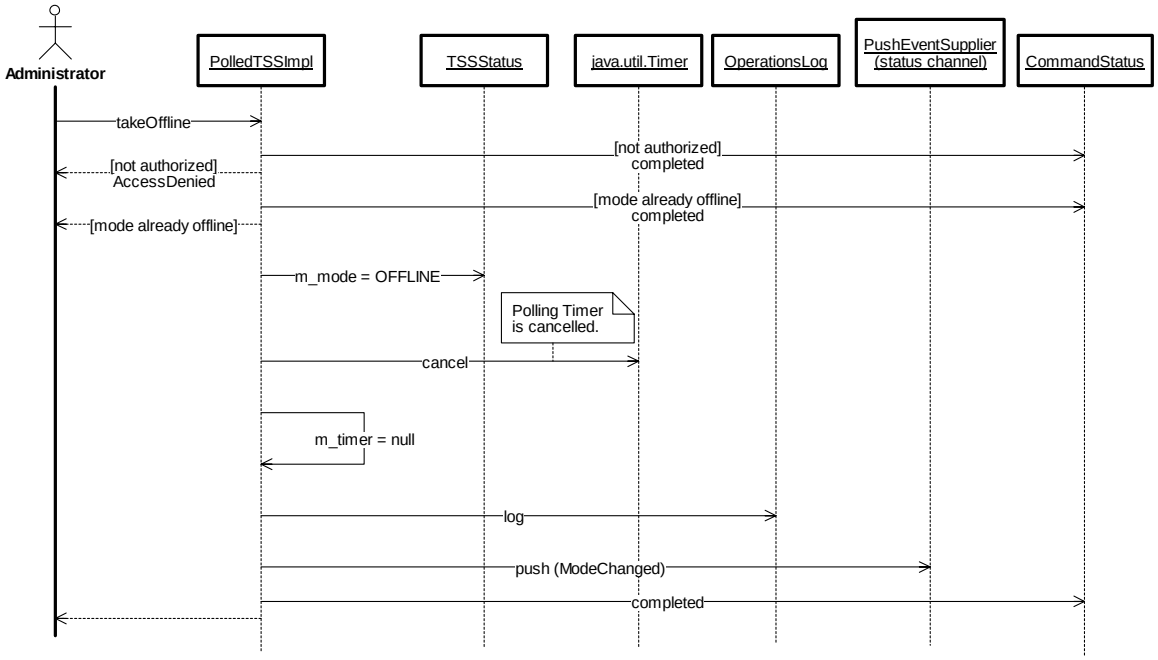

Figure 8. PolledTSSImpl:takeOffline (Sequence Diagram)

3.2.4.5 RTMSFactoryImpl:constructor (Sequence Diagram)

When the RTMSFactoryImpl is constructed, it obtains persisted data for each previously existing RTMS from the database and constructs RTMSImpl objects using this data. Each object is connected to the ORB and registered in the CORBA trading service. The factory creates a timer that is used to cause it to periodically collect the status of all RTMS objects and push the data as a CORBA event.

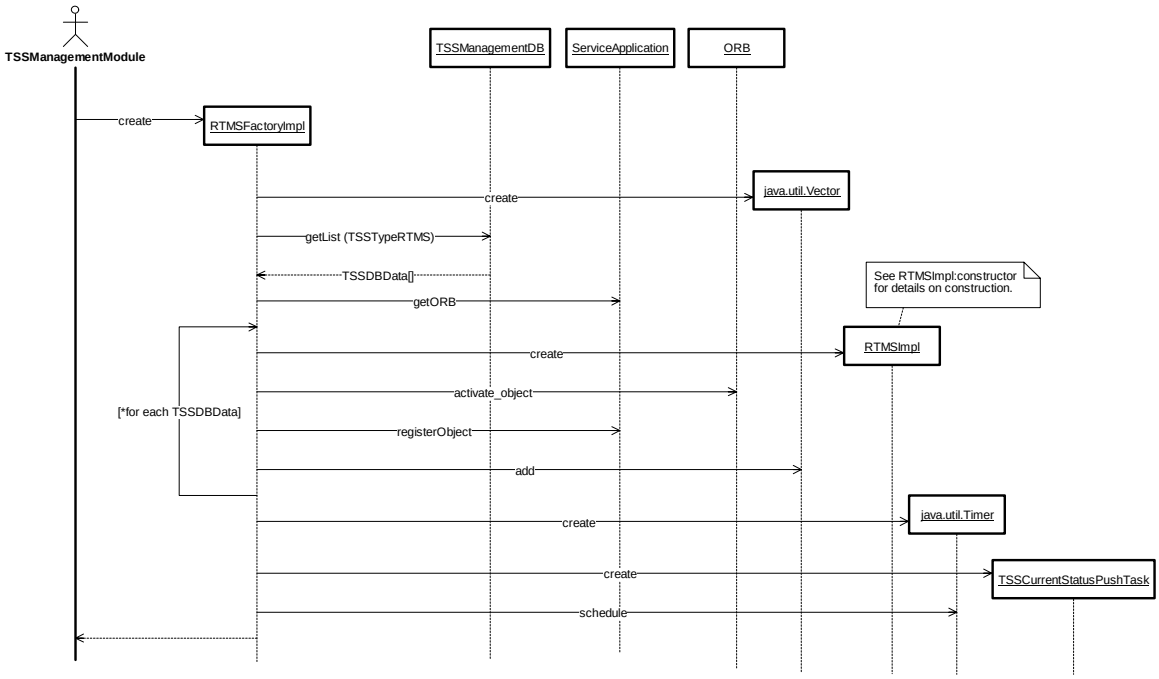

Figure 9. RTMSFactoryImpl:constructor (Sequence Diagram)

3.2.4.6 RTMSFactoryImpl:createRTMS (Sequence Diagram)

A user with the proper functional rights can add an RTMS to the system. The RTMSFactoryImpl is called with configuration data for the RTMS to be added. The RTMSFactoryImpl adds the configuration data to the database, using status information indicating the device is offline and OK. An RTMSImpl object is created using this same data and the object is added to the list of RTMSImpl objects managed by the factory. The new RTMSImpl object is connected to the ORB and published in the CORBA Trading Service. A CORBA event is pushed to allow other applications to be notified of the existence of the RTMS object.

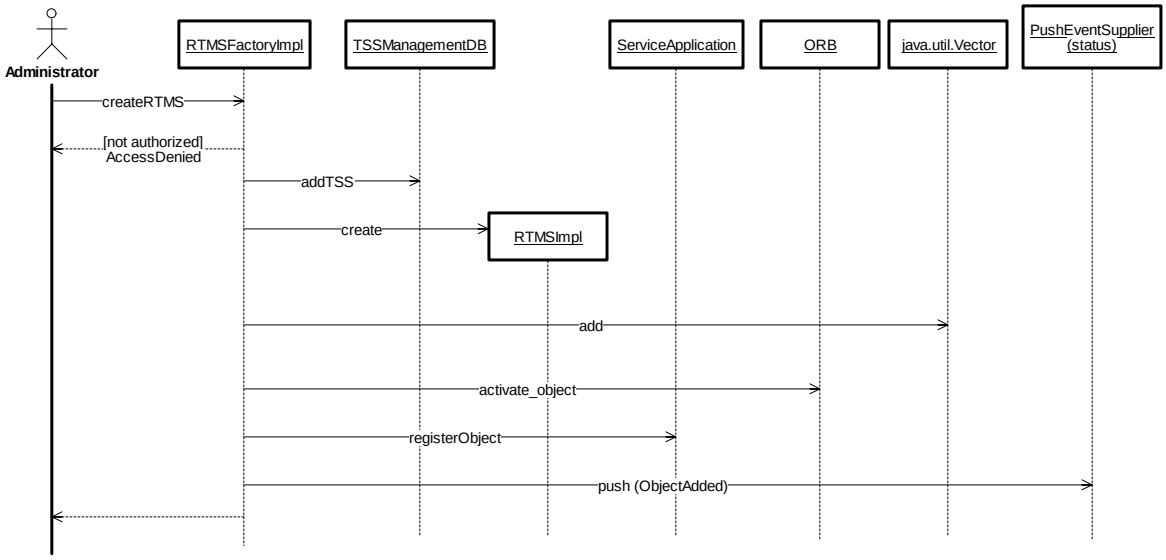

Figure 10. RTMSFactoryImpl:createRTMS (Sequence Diagram)

3.2.4.7 RTMSFactoryImpl:CurrentStatusPush (Sequence Diagram)

The RTMSFactoryImpl contains a timer used to periodically push the current status of all sensors managed by the factory. The factory retrieves the status of each RTMS and bundles all status into a single CORBA event. This event is pushed on the Data event channel.

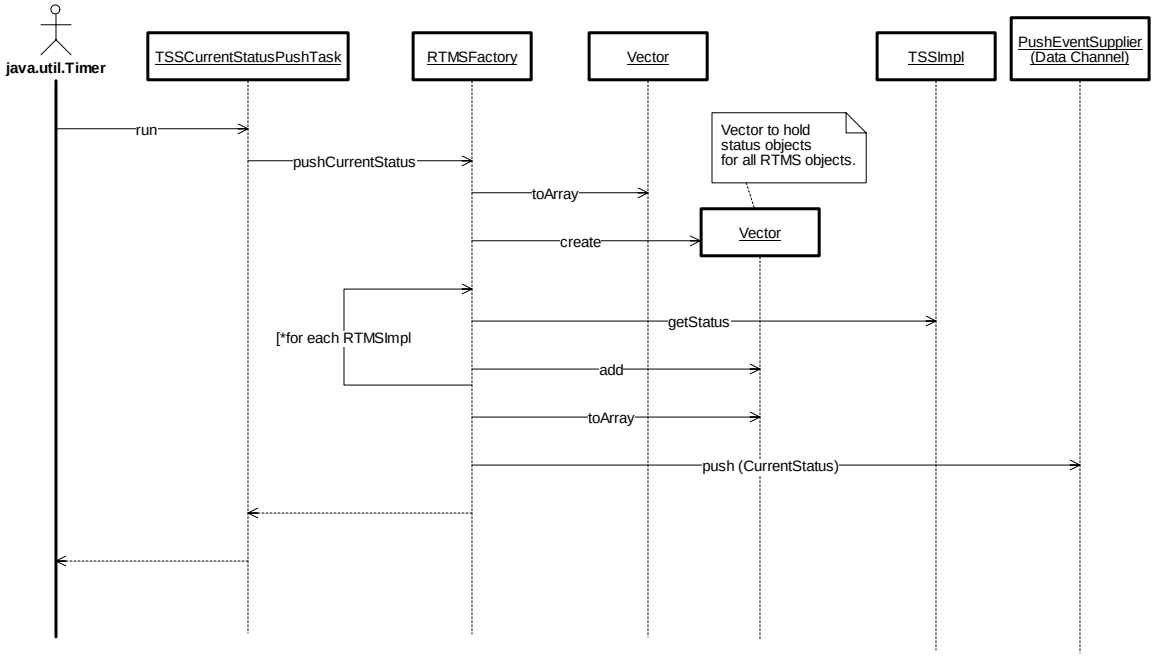

Figure 11. RTMSFactoryImpl:CurrentStatusPush (Sequence Diagram)

3.2.4.8 RTMSFactoryImpl:remove (Sequence Diagram)

A user with the proper functional rights can remove an RTMS from the system. The RTMSFactory withdraws the object from the CORBA trading service, disconnects the object from the ORB, removes the object's persisted data from the database, and finally removes the object from the factory's list of RTMS objects. A CORBA event is pushed to notify other applications of the object's removal.

Note that this diagram shows an object being removed through a direct call to the RTMSFactoryImpl. RTMS objects can also be removed using the remove method of the RTMS object. When this occurs, the RTMS object simply delegates the call to its factory and the processing occurs as if the factory was called directly.

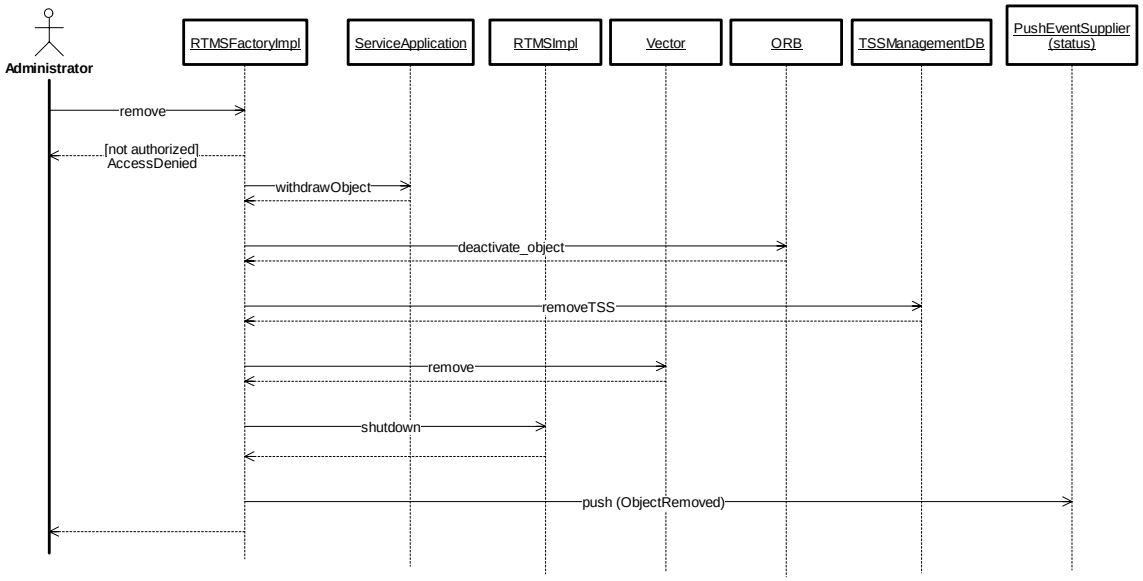

Figure 12. RTMSFactoryImpl:remove (Sequence Diagram)

3.2.4.9 RTMSImpl:constructor (Sequence Diagram)

This diagram shows the construction of the RTMSImpl object. The RTMSImpl invokes the base class constructor, allowing it to construct a PortLocator, LogFile (for debugging), and a polling timer (if the status passed to the constructor does not indicate the device is offline). After the base class is constructed, the RTMSImpl constructs an RTMSProtocolHandler to be used to perform the RTMS specific protocol to obtain traffic parameters from the RTMS device.

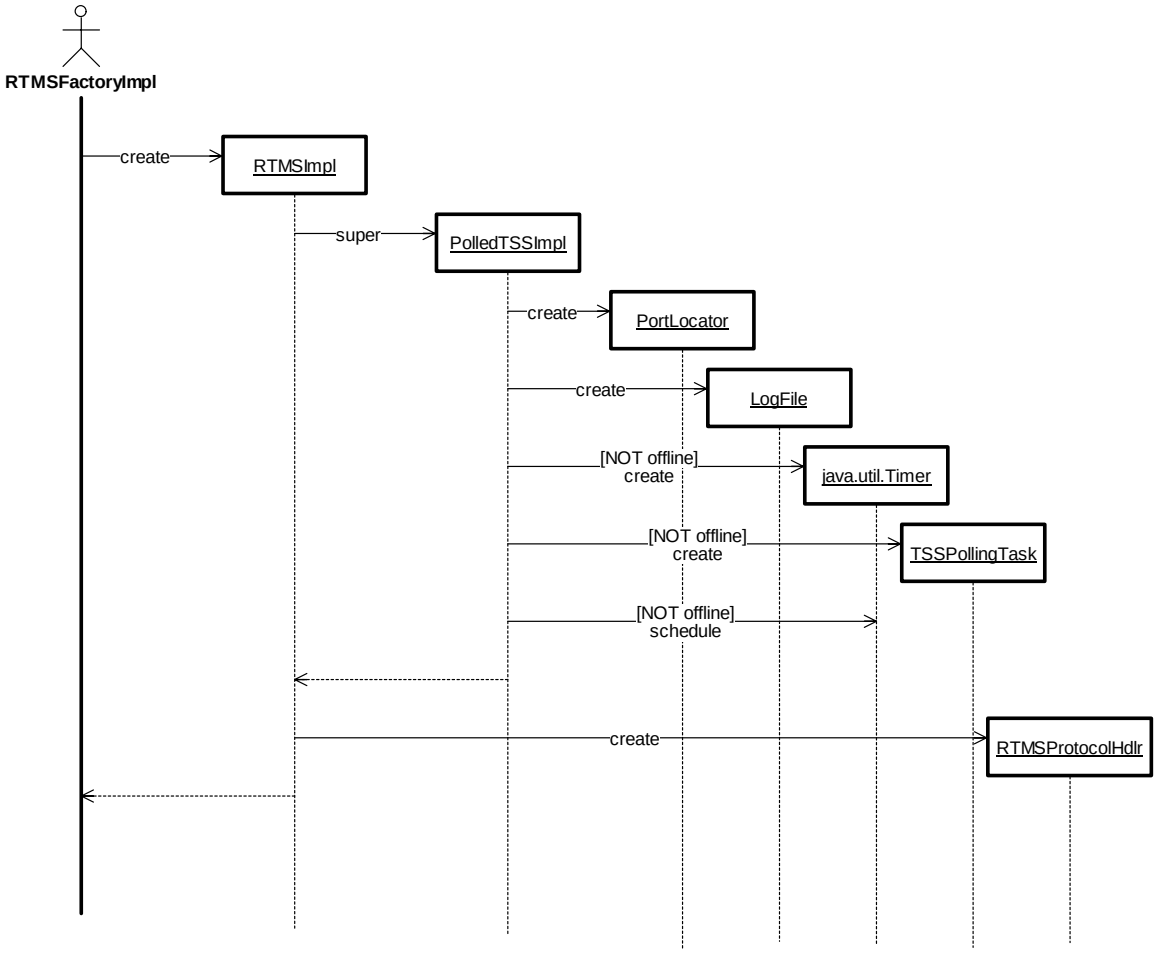

Figure 13. RTMSImpl:constructor (Sequence Diagram)

3.2.4.10 RTMSImpl:poll (Sequence Diagram)

The poll method of the RTMSImpl is called from its base class when it is time to poll the RTMS device. At the point when this method is called, the base class has already established a connection with the device. The RTMSImpl uses the RTMSProtocolHandler to send a data request to the device and parse the device response. Any communication failure, such as a non-responsive device, causes the base class to be notified that a communication failure occurred. If a communication failure did not occur, the RTMS health status is checked for an indication of a hardware failure. If no hardware failure exists, the lane level data is passed back to the base class to process the data.

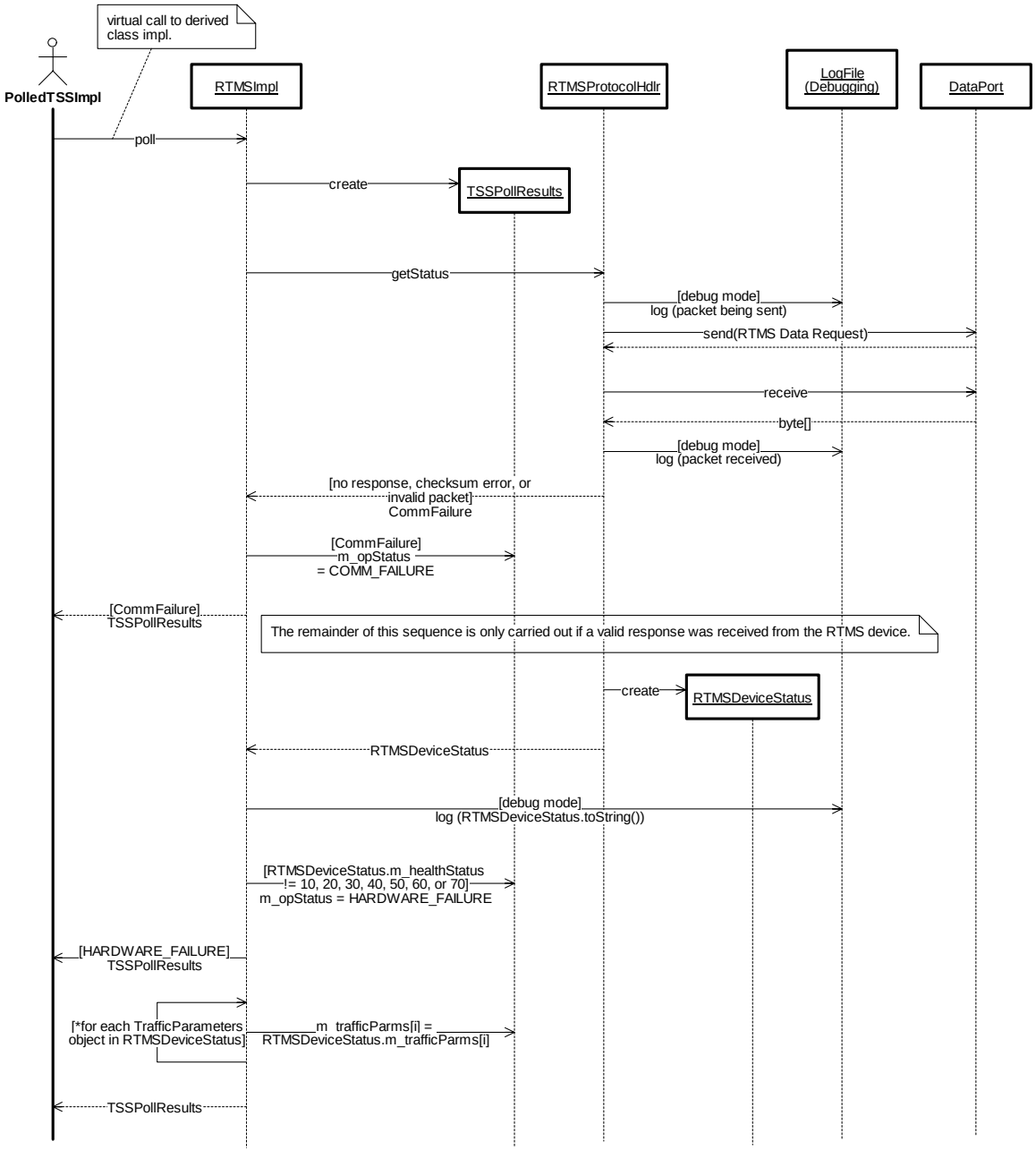

Figure 14. RTMSImpl:poll (Sequence Diagram)

3.2.4.11 RTMSImpl:remove (Sequence Diagram)

A user with the proper functional rights can remove an RTMS from the system. When this is done through a call to the RTMS object, the RTMS delegates the call to the RTMS Factory. See the RTMSFactoryImpl:remove sequence for details.

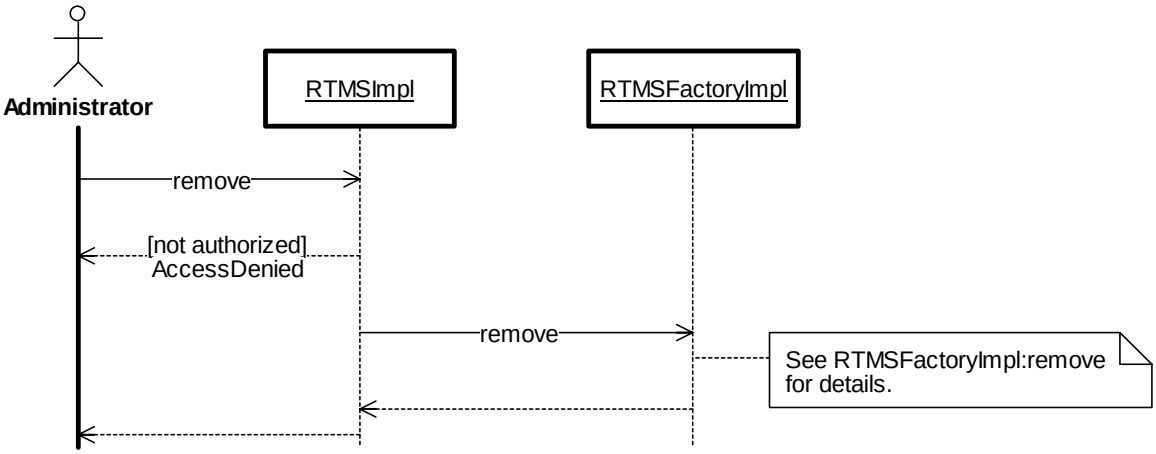

Figure 15. RTMSImpl:remove (Sequence Diagram)

3.2.4.12 TSSManagementModule:initialize (Sequence Diagram)

The TSSManagementModule is initialized by the ServiceApplication framework when the Chart2Service configured to contain the TSSManagementModule is started. The TSSManagementModule first ensures that the proper offer types have been registered in the Trader for the types of objects this module will serve. It creates a wrapper to the service's properties object that provides easy access to properties that are specific to this module. A TSSManagementDB object is created to provide access to Transportation Sensor System data that is stored in the database. Two PushEventSupplier objects are created to provide access to two separate CORBA event channels. One event channel is used to push events relating to configuration and operational status of RTMS objects. The other channel is used to periodically push traffic parameter data to interested parties. A LogFile object is created to provide access to a raw data log file, used to log lane level data to a flat file. Finally, the RTMSFactoryImpl object is created, connected to the ORB, and registered in the CORBA Trading Service.

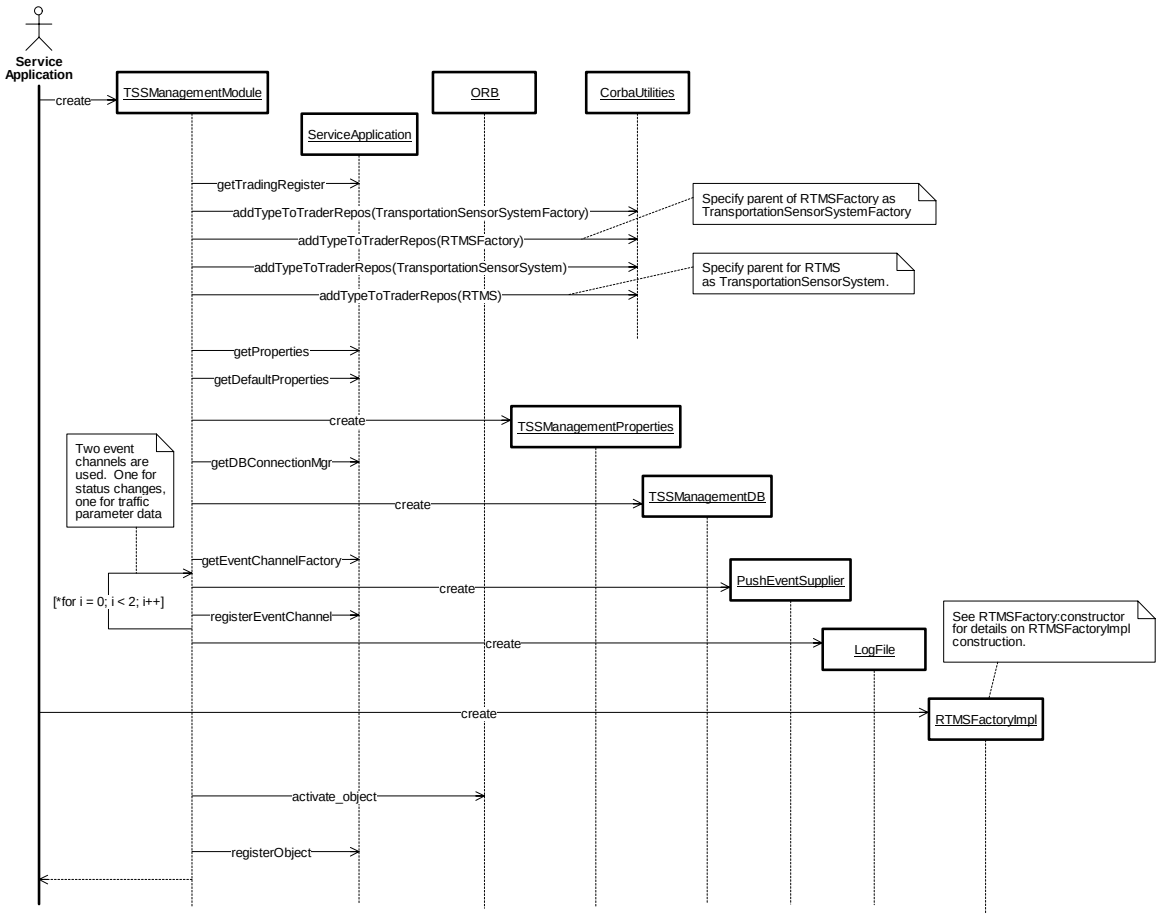

Figure 16. TSSManagementModule:initialize (Sequence Diagram)

3.2.4.13 TSSManagementModule:shutdown (Sequence Diagram)

The TSSManagementModule is shutdown when the Chart2Service that contains the module is shut down. The TSSManagementModule disconnects the RTMSFactory from the ORB and then tells it to shut down. The RTMSFactoryImpl tells each RTMSImpl object to shut down and it disconnects the object from the ORB. When an RTMSImpl object is shut down, it cancels its polling timer (if any).

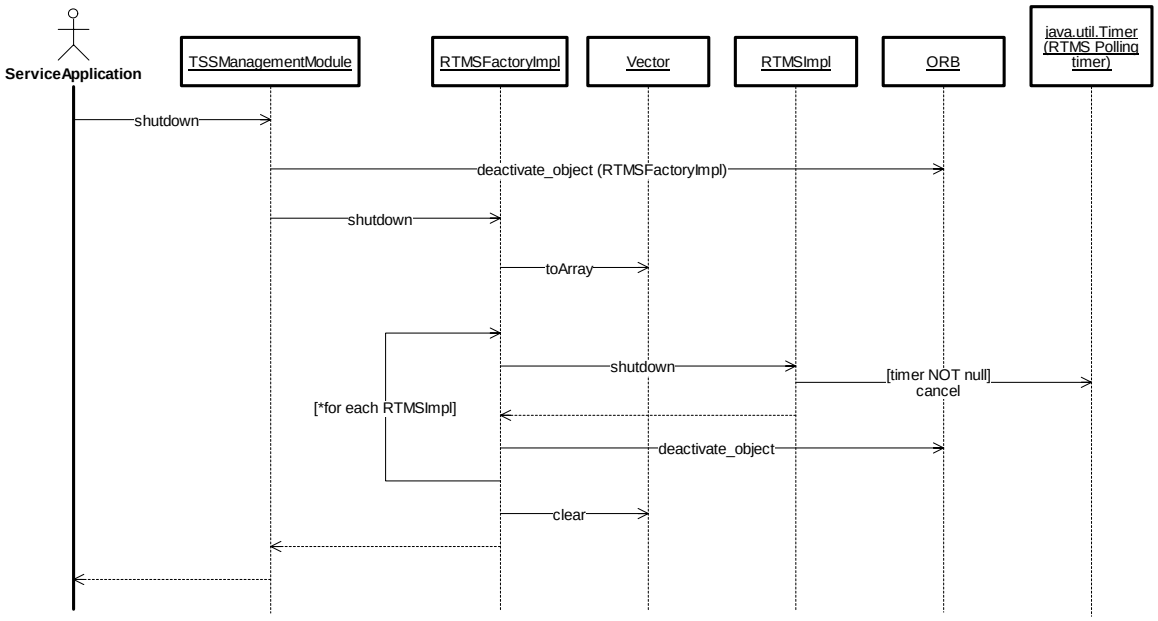

Figure 17. TSSManagementModule:shutdown (Sequence Diagram)

3.2.4.14 TSSPollingTask:run (Sequence Diagram)

A PolledTSSImpl object is polled on a regular interval specified in its TSSConfiguration object. When polled, the PolledTSSImpl returns immediately if it is offline. Otherwise, it establishes a connection with the field device using communications parameters specified in the TSSConfiguration. The PolledTSSImpl then calls its abstract poll method that is implemented by the derived class (RTMSImpl). Any changes to the operational status detected during polling of the device are pushed on the CORBA event channel used for status. Raw data obtained from the device is logged in the Raw Data log file. The lane level data provided by the derived class is combined according to the zone groups that have been configured in the TSSConfiguration. The speed for each lane (detection zone) included in a zone group is averaged to provide a single value for the zone group. If a lane does not have any volume (and thus no speed), the speed obtained the last time the lane had a volume is used. The volume across all detection zones in the zone group is summed, and the occupancy for all detection zones in the zone group is averaged (including detection zones reporting zero occupancy). The current speed for each detection zone with a non-zero volume is stored off for use when a zero occupancy occurs.

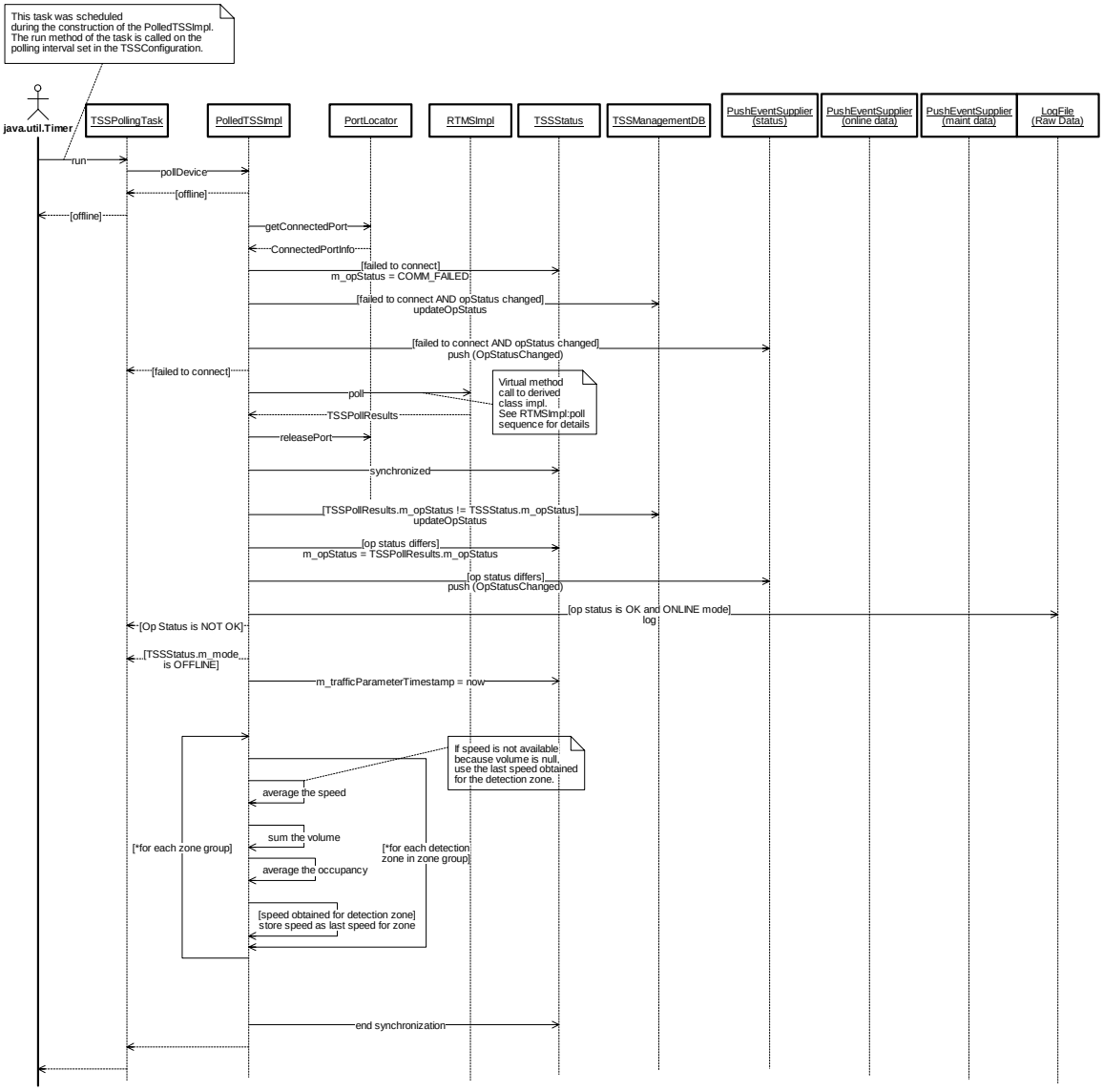

Figure 18. TSSPollingTask:run (Sequence Diagram)

3.3 GUITSSModule

3.3.1 GUITSSModuleClasses (Class Diagram)

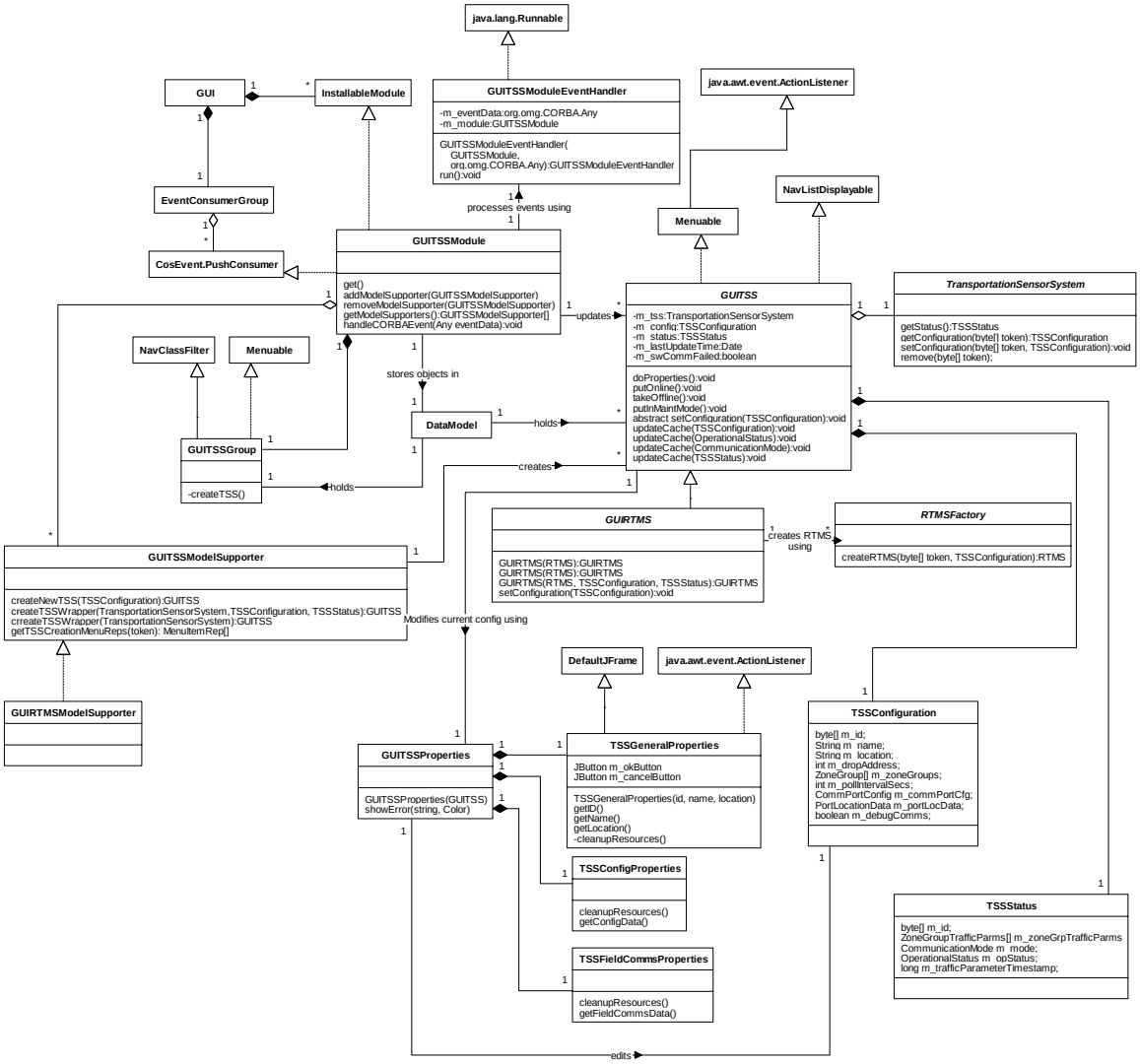

Figure 19. GUITSSModuleClasses (Class Diagram)

3.3.1.1 CosEvent.PushConsumer (Class)

The PushConsumer interface is the interface to an event channel that a supplier of information uses to push event updates to consumers who have previously attached to the channel.

3.3.1.2 DataModel (Class)

The data model class serves as a collection of objects. It provides an efficient lookup mechanism for locating any object, and methods that allow for the retrieval of all objects of a particular type. Additionally, this class provides the ability to attach observer objects that are notified when objects are added to or removed from the model. Objects may also notify the DataModel that they have been modified. The model will periodically notify all attached observers of the changes to objects in the model.

3.3.1.3 DefaultJFrame (Class)

This class provides a default implementation of the WindowManageable interface, and may be used as a base class for other frame windows in the GUI. It handles all interactions with the WindowManager for attaching and detaching, as well as saving the window position.

3.3.1.4 EventConsumerGroup (Class)

This class represents a collection of event consumers, which will be monitored to verify that they do not lose their connection to the CORBA event service. The class will periodically ask each consumer to verify its connection to the event channel on which it is dependant to receive events.

3.3.1.5 GUI (Class)

This class is a singleton that contains all of the centralized functionality in the GUI. This includes startup, shutdown, login, and logout. It manages the installable modules and controls all functionality, which requires the modules to be called. In addition, it stores all of the CORBA object wrappers in the DataModel, which allows access to the objects and supports an update mechanism to notify interested observers whenever the objects change.

3.3.1.6 GUIRTMS (Class)

This class is a GUITSS subclass that contains code that is specific to the RTMS. This class can be constructed in one of three ways. The default constructor creates an empty wrapper object, to be used during the addition of an RTMS to the system. A second constructor exists that takes the object reference to be wrapped as a parameter. This method is used during discovery and must use the object reference to obtain the current configuration and status information from the object being wrapped. The third constructor, used to process an ObjectAdded event, constructs the wrapper with configuration and status data that is already known (because it is pushed in the ObjectAdded event).

This class implements the abstract setConfiguration() method defined in its base class. The implementation behaves differently based on whether the wrapper is currently empty. If the wrapper is not empty, the method serves to change the configuration on an existing RTMS object. If the wrapper is not empty, the method serves to create a new RTMS object using the given configuration.

3.3.1.7 GUIRTMSModelSupporter (Class)

This class is an implementation of GUITSSModelSupporter that provides support for RTMS devices. This class can provide a menu item for the TSS folder's context menu to allow the user to add an RTMS object to the system. This class can also create a GUI wrapper for an existing RTMS or create an empty wrapper for an RTMS that is in the process of being added to the system.

3.3.1.8 GUITSS (Class)

This class is a wrapper object for a TransportationSensorSystem object that exists within the CHART II system (on the server). This object serves to cache configuration and status data for the wrapped object to eliminate the need for most CORBA calls to the wrapped object. This object obtains initial values for the configuration and status during the GUI object discovery or through an ObjectAdded event pushed on a CORBA event channel. After the initial configuration and status is obtained, this object keeps its cached data current through calls to the updateCache methods, which are called by the GUITSSModule as it receives asynchronous CORBA events pushed from the server when configuration or status changes.

Although this class contains common processing that can be used for any TransportationSensorSystem, it is abstract and must be subclassed to provide an implementation of the setConfiguration method. The setConfiguration method is called when the configuration values for a TSS are being changed by the user or when the configuration values have been entered for the first time to add a TSS to the system. In the former case, the derived class must simply pass the changed configuration to the wrapped object's setConfiguration method. In the latter case, however, the derived class must seek out a factory capable of creating the specific type of TSS being added and have the factory add the TSS to the system.

3.3.1.9 GUITSSGroup (Class)

This object is the root TSS folder that appears in the navigator tree. This object inherits much of its functionality from the NavClassFilter class, but provides custom implementations of the getSSMenuItemReps method (which is used to construct a context menu for the TSS folder) and the getAllNavProperties method, which is used to determine the columns to display in the navigator when the user selects the TSS foler in the navigator tree. This class is also an ActionListener. Its actionPerformed method is called when the user selects any of the menu items contained in the TSS folder's pop-up menu.

3.3.1.10 GUITSSModelSupporter (Class)

This interface is implemented by classes that can provide model specific services, such as creating a new Transportation Sensor System GUI wrapper.

The createNewTSS method is called when a TSS object is in the process of being added to the system. The implementing class creates an empty GUITSS derived object and passes the object back to the caller.

The createTSSWrapper method is called when a TSS that exists in the system has been discovered, either via object discovery or through notification of an ObjectAdded via an event channel. The implementing class caches the current configuration and status, using the parameters passed or by going to the wrapped object and retrieving the data.

The getTSSCreationMenuReps method is used to allow the implementing class to add a "create" menu item(s) for the type(s) of TSS the implementing class supports. An example of such a menu item would be one with text "Add RTMS", which would be added by the GUIRTMSModelSupporter.

3.3.1.11 GUITSSModule (Class)

This class is an installable GUI module that provides GUI support for the display and configuration of Transportation Sensor System (TSS) devices. It is responsible for ensuring the TSS root appears in the navigator tree as well as any filters that are created as children of the TSS root. It implements the CORBA PushConsumer interface and is therefore a CORBA object that is connected to the ORB and is called remotely by an EventChannel (via its PushConsumer push() method) when data is pushed on the channel. This module connects to the TSS status channels that exist in the system to allow this module to be notified of status and configuration changes to TSS objects. (Note the TSS status channel does not push traffic parameters that have been read from the device).

3.3.1.12 GUITSSModuleEventHandler (Class)

This class is used to cause the GUITSSModule to process CORBA events to be processed on the main GUI thread to ensure a consistent state for GUI wrapper objects. This object is constructed holding the event data that was received and a reference to the GUITSSModule. It is then passed to the SwingUtilities to be invoked asynchonously on the main GUI thread. When invoked, this object calls the handleCORBAEvent method in the GUITSSModule.

3.3.1.12.1 java.lang.Runnable (Class)

This interface allows the run method to be called from another thread using Java's threading mechanism.

3.3.1.13 GUITSSProperties (Class)

This class is the window that is shown when the user clicks on a GUITSS object's Properties menu or when the user is adding a TSS to the system. This window contains a tabbed pane used to show the TSS configuration values grouped into logical categories on separate tabs. The TSSGeneralProperties, TSSConfigProperties, and TSSFieldCommsProperties are all windows that appear on this window's tabbed pane and contain various GUI controls to allow the configuration values of a TSS to be viewed and/or changed.

3.3.1.14 InstallableModule (Class)

This class is the basic interface that all installable modules must implement. It contains functionality that all modules must support to be installable modules. This includes functionality for startup, shutdown, login, logout, and the handling of system and user preferences.

3.3.1.15 java.awt.event.ActionListener (Class)

This interface listens for actions such as when a menu item or button is clicked. For menu items, it is attached to menu items when the menu is built.

3.3.1.16 Menuable (Class)

This interface allows an object to provide menu item strings and receive commands when the corresponding menu items are clicked on. It supports both single selection and multiple selection of Menuable objects. The getSSMenuItems() method should return the menu items to display if the object is singly selected. The getMSMenuItems() method should return the menu items that the Menuable object wishes to display if other Menuable objects are selected. The access token is passed to these methods to allow the Menuable object to check the user's access rights before supplying the strings, so the user's actions may be restricted.

3.3.1.17 NavClassFilter (Class)

This filter ignores all objects that are not assignable to a given class or interface. Thus, an interface or base class can be specified and all of the objects implementing the interface or extending the base class will be included.

3.3.1.18 NavListDisplayable (Class)

This interface must be implemented by any object to be displayed on the right hand side of the Navigator window, in the list view. In addition to the Navigable methods, it must also support getting and comparing the strings for a given property (column) in the list.

3.3.1.19 RTMSFactory (Class)

Objects which implement RTMSFactory are capable of adding an RTMS to the system.

3.3.1.20 TransportationSensorSystem (Class)

A Transportation Sensor System (TSS) is a generic term used to describe a class of technology used for detection within the transportation industry. Examples of TSS devices range from the advanced devices, such as RTMS, to basic devices, such as single loop detectors.

This software interface is implemented by objects that provide access to the traffic parameters sensed by a Transportation Sensor System. Transportation Sensor Systems are capable of providing detection for one or more detection zones. A single loop detector would have one detection zone, while an RTMS would have eight detection zones.

3.3.1.21 TSSConfigProperties (Class)

This class is a window that allows the configuration of zone groups for a TSS to be specified / changed. Zone groups can be added, removed, or modified. Each zone group can be configured to have an optional description, a direction, and one or more detection zone numbers.

3.3.1.22 TSSConfiguration (Class)

This class holds configuration data for a transportation sensor system (TSS) as follows:

m_id — The unique identifier for this TSS. This field is ignored when the object is passed to the TSS to change its configuration.

m_name — The name used to identify the TSS.

m_location — A descriptive location of the TSS.

m_dropAddress — The drop address for the device.

m_zoneGroups — Logical groupings of detection zones, used to provide a single set of traffic parameters for one or more detection zones.

m_pollIntervalSecs — The interval on which the TSS should be polled for its current traffic parameters (in seconds).

m_commPortCfg — Communication configuration values.

m_portLocData — Configuration information that determines which port manager(s) should be used to establish a connection with the SensorSystem.

m_debugComms — Flag used to enable/disable the logging of communications data for this TSS. When enabled, command and response packets exchanged with the device are logged to a debugging log file.

3.3.1.23 TSSFieldCommsProperties (Class)

This class is a window that allows communication related configuration values to be set. This includes the drop address for the device, the polling interval, and comm port settings such as baud rate, databits, parity, etc. The window also allows the setting of a default phone number, the modem type (ISDN or POTS) to use to access the device, and one or more port managers to be specified as the origin for field communications, each with its own version of the default phone number. Lastly, a debug flag can be set to cause the data packets exchanged with the device to be logged in a debug file.

3.3.1.24 TSSGeneralProperties (Class)

This class is a window that contains GUI controls to allow general configuration values of a TSS to be viewed and modified. Controls exist to allow the name and location of a TSS to be specified.

3.3.1.25 TSSStatus (Class)

This class holds current status information for a TSS as follows:

m_id — The ID of the TSS for which this status applies.

m_zoneGrpTrafficParms — The traffic parameters for each ZoneGroup of the Transporation Sensor System as specified in the Sensor system's TSSConfiguration object.

m_mode — The communication mode of the TSS.

m_opStatus — The operational status for the TSS.

m_trafficParameterTimestamp — A timestamp that records when the traffic parameter data was collected from the device.

3.3.2 Sequence Diagrams

3.3.2.1 GUIRTMSModelSupporter:getTSSCreationMenuReps (Sequence Diagram)

As part of processing a right click on the TSS group in the navigator tree, the GUITSSGroup allows each model supporter to contribute to the context menu. When the GUIRTMSModelSupporter is asked for its creation menu items, it returns an array with a single menu item, “Add RTMS.” This menu item is disabled if the user does not have configure TSS privileges.

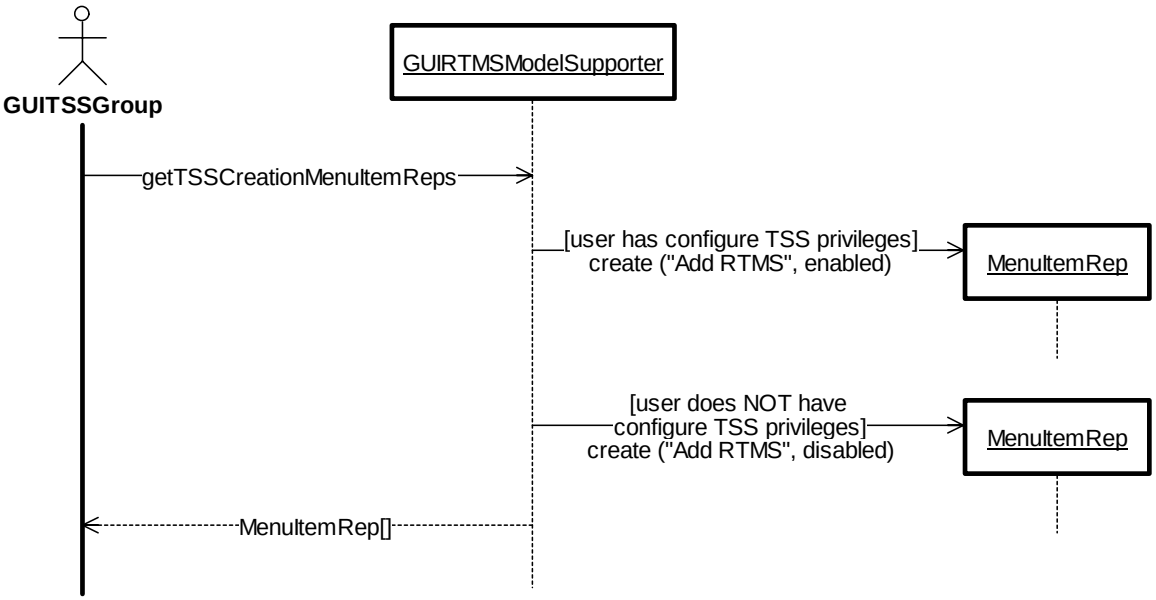

Figure 20. GUIRTMSModelSupporter:getTSSCreationMenuReps (Sequence Diagram)

3.3.2.2 GUIRTMSModelSupporter:createTSSWrapper (Sequence Diagram)

The GUIRTMSModelWrapper createTSSWrapper method is called when the GUITSSModule discovers a TSS in the system via a trader query or when an event is received indicating a TSS has been added to the system. If the TSS that has been discovered or added is not an instance of an RTMS, this method returns null, indicating this model supporter does not support the type of TSS that has been discovered or added. If the TSS discovered or added is an RTMS, a wrapper object is created. In the case of an RTMS that has been discovered, the current configuration and status of the RTMS is not known and the wrapper retrieves this information from the RTMS object that is being wrapped.

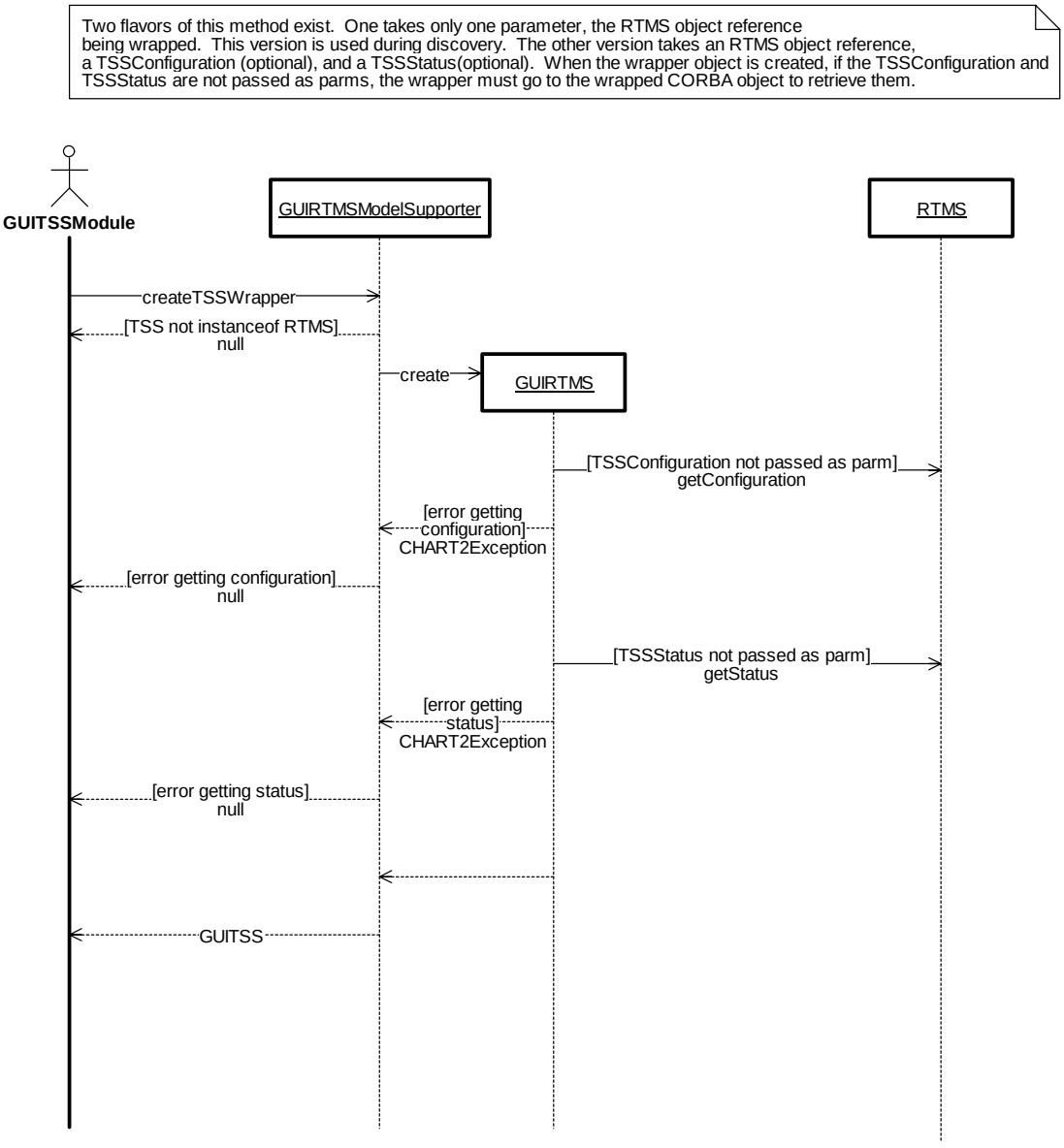

Figure 21. GUIRTMSModelSupporter:createTSSWrapper (Sequence Diagram)

3.3.2.3 GUIRTMS:setConfiguration (Sequence Diagram)

When the OK button is pressed on the GUITSSProperties window, it calls the setConfiguration() method on the GUITSS object which spawned the window. The GUIRTMS subclass of GUITSS implementation of the setConfiguration() method contains processing specific to setting the configuration of an RTMS. If a reference to a TransportationSensorSystem already exists in the GUIRTMS object, the setConfiguration() method of the TransportationSensorSystem is called. If a reference does not exist, this indicates the object was newly created in the GUI and needs to be added to the system. The GUIRTMS accomplishes this by locating an RTMSFactory object in the system and attempting to have the factory create a new RTMS object. Because there may be more than one factory in the system capable of creating RTMS objects, this method tries to create an RTMS object on each known factory until it achieves success or all factories have been tried once or an AccessDenied exception is caught.

Note that when an RTMS is added to the system, its wrapper object (GUIRTMS) does not get added to GUI within this routine. Instead, the wrapper is added to the GUI in the event processing done on the CORBA event that is pushed from the RTMSFactory when the RTMS is added. This allows the GUI to generically handle the addition of RTMSs, regardless of whether the GUI happens to be the instance where the command to add the RTMS originated.

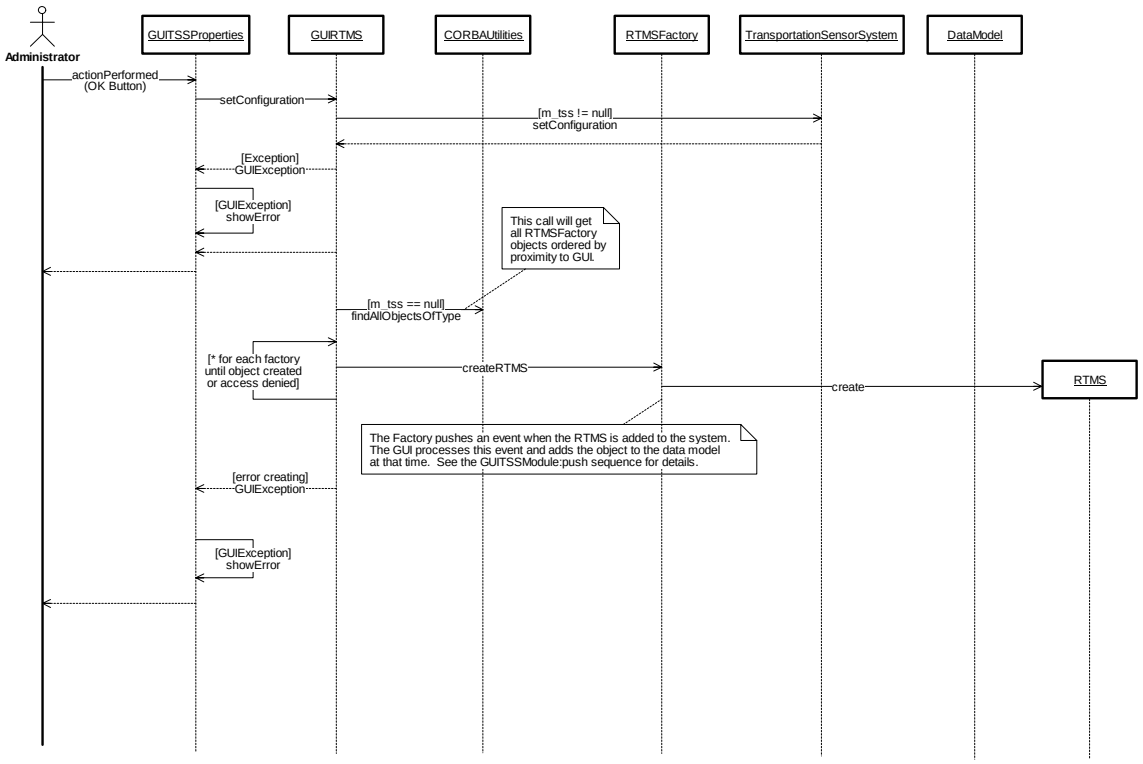

Figure 22. GUIRTMS:setConfiguration (Sequence Diagram)

3.3.2.4 GUITSS:actionPerformed (Sequence Diagram)

When a user selects a menu item on a GUITSS context menu, the GUITSS actionPerformed method is called. The GUITSS actionPerformed method simply routes the command to the appropriate method for processing.

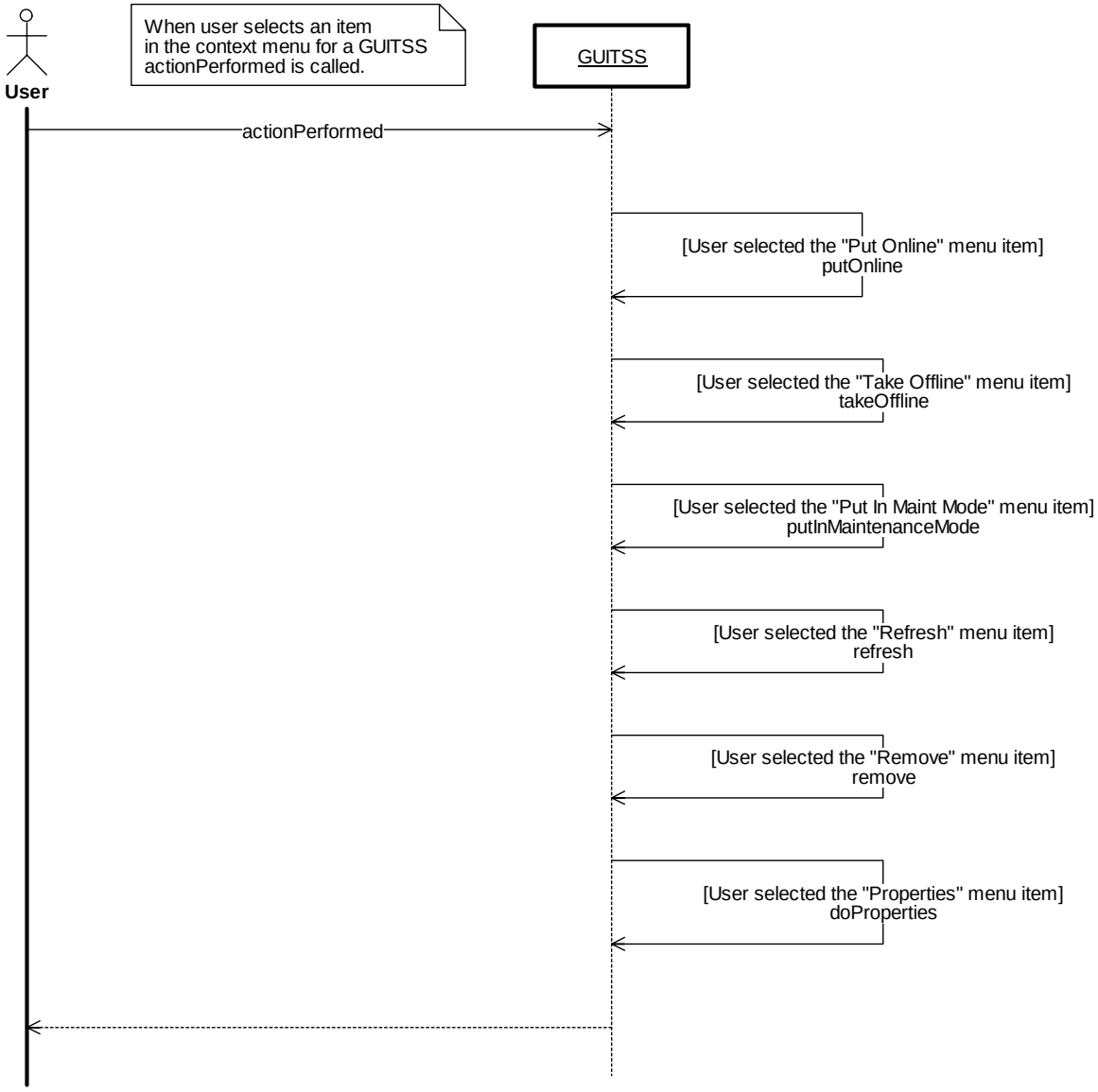

Figure 23. GUITSS:actionPerformed (Sequence Diagram)

3.3.2.5 GUITSS:allowSetDesc (Sequence Diagram)

The navigator uses the allowSetDesc method to determine if an object displayed in the navigator allows in-place editing of the description field. The GUITSS does not allow in-place editing of the description field and always returns false.

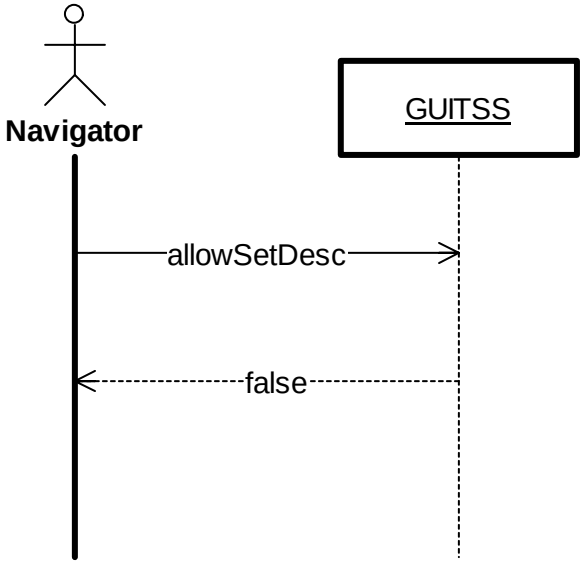

Figure 24. GUITSS:allowSetDesc (Sequence Diagram)

3.3.2.6 GUITSS:compareProperty Values (Sequence Diagram)

The navigator calls an object's compareProperty Values method to determine ordering of objects within the navigator when sorting occurs. The navigator by default sorts by doing a string comparison. The GUITSS overrides this standard sorting for the Comm Mode, Op Status, and last update time columns. The last update time is sorted as expected, with later times being greater than earlier times. The comm mode is sorted in ascending order as online, offline, maint mode. The op status is sorted as Hardware Failed, SW Comm Failed, Comm Failed, and OK.

Note that the naviagator toggles sorts between ascending and descending order.

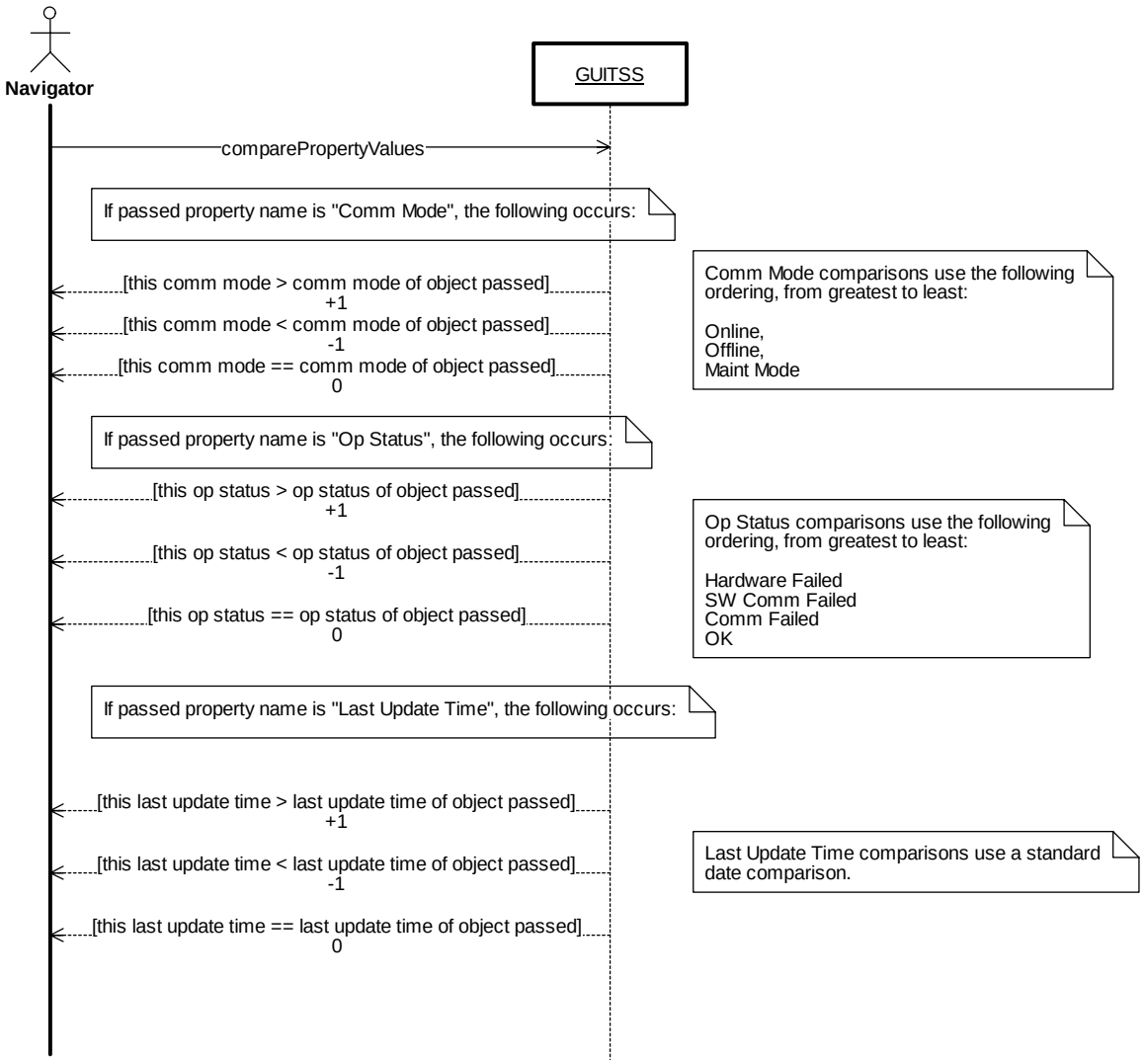

Figure 25. GUITSS:compareProperty Values (Sequence Diagram)

3.3.2.7 GUITSS:doProperties (Sequence Diagram)

The GUITSS doProperties method is called by the GUITSS when a user selects the "Properties" menu item from the GUITSS context menu or when a TSS is being added to the system. The GUITSS creates the GUITSSProperties window that in turn creates windows for its tabbed pane. The GUITSS then shows the window and returns. At this point the user interacts with the window, viewing or changing values and pressing OK or cancel when they are finished. When the cancel button is pressed, the window is dismissed, making no changes to the TSS configuration and without adding a TSS to the system. When the OK button is pressed, data is collected from the GUI components on the properties windows and the GUITSS setConfiguration method is called. If the setConfiguration method is successful, the properties window is dismissed. If an error occurs, the error is displayed in an error panel and the properties window remains open.

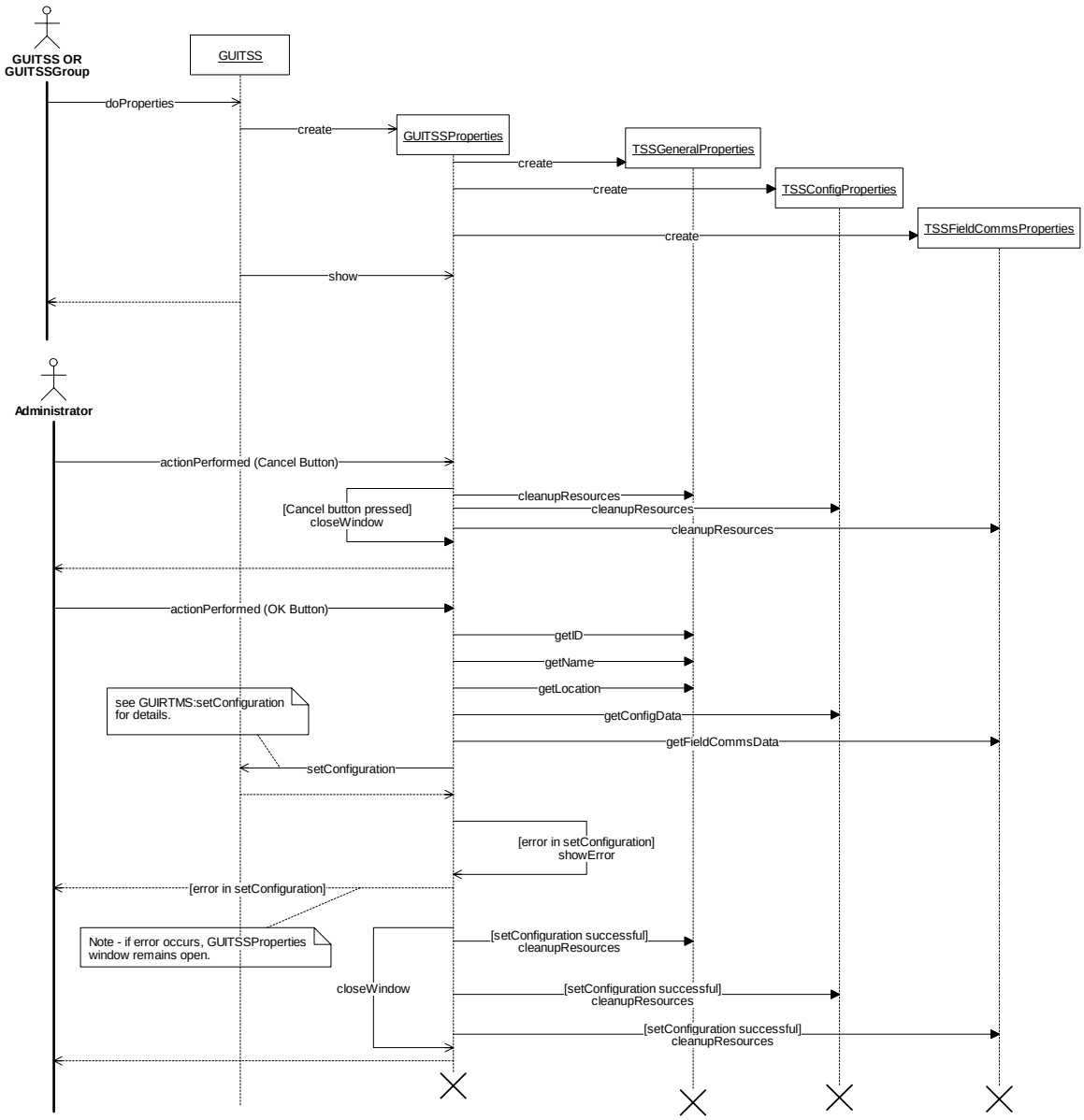

Figure 26. GUITSS:doProperties (Sequence Diagram)

3.3.2.8 GUITSS:getDesc (Sequence Diagram)

When the navigator asks the GUITSS for its description, the GUITSS simply returns the name of the device as stored in its cached TSSConfiguration object.

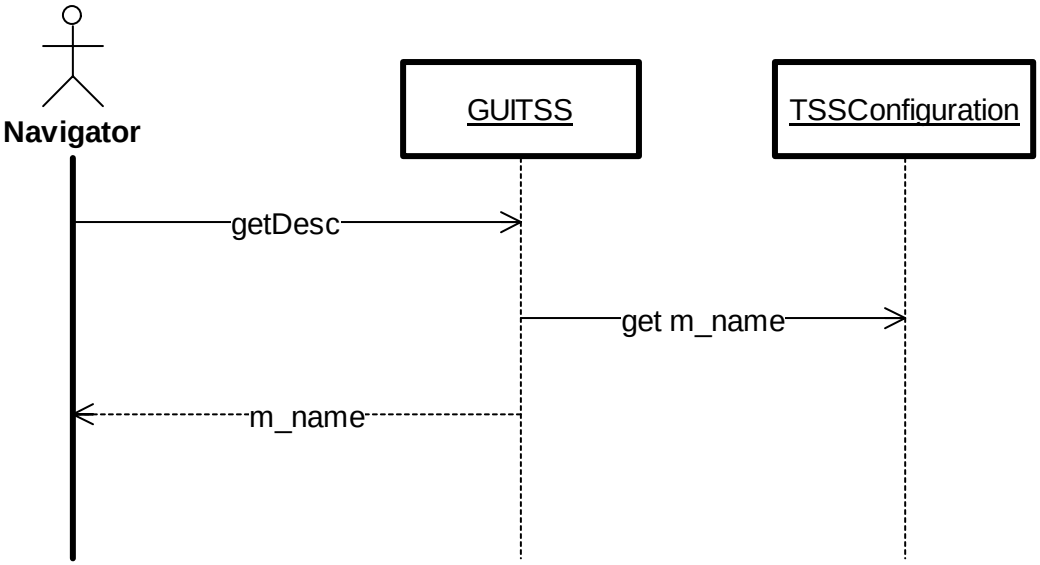

Figure 27. GUITSS:getDesc (Sequence Diagram)

3.3.2.9 GUITSS:getImage (Sequence Diagram)

When the navigator displays a row for a TSS, it asks the GUITSS for the image to display in the first column of the navigator. The GUITSS returns a pre-loaded image based on its current communication mode and operational mode. Note that an operational mode of software comm failed exists on the GUI but not on the server side object. This state is used to represent a condition where the GUI is unable to reach the server side object.

The GUITSS pre-loads all images into static variables when the class is first loaded through the use of a static block. This allows all instances of GUITSS objects to share the same image objects.

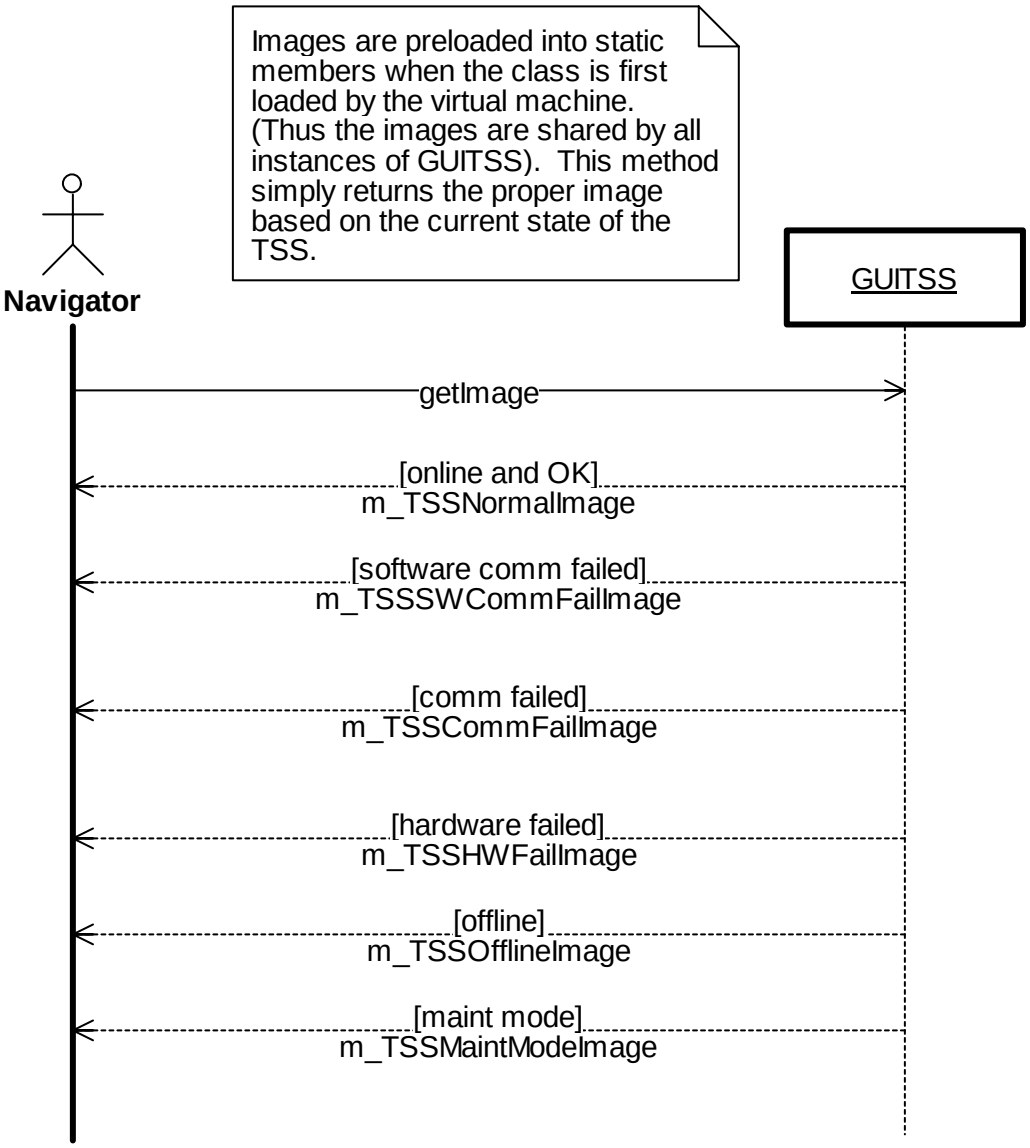

Figure 28. GUITSS:getImage (Sequence Diagram)

3.3.2.10 GUITSS:getMSMenuItemReps (Sequence Diagram)

When a user selects multiple TSS objects in the navigator and then right clicks on one, the navigator asks each GUITSS object for its list of menu items to appear in the context menu. The navigator displays only the menu items that are returned from all selected items. The possible menu items to appear in a TSS pop-up menu are as follows:

Properties — only visible if device is in maintenance mode. Disabled if user does not have TSS configuration privileges.

Put Online — only visible if device is not already online. Disabled if in maint mode and user does not have maint mode privileges. Disabled if offline and user does not have device comms privileges.

Take Offline — only visible if device is not already offline. Disabled if in maint mode and user does not have maint mode privelges. Disabled if online and user does not have device comms privileges.

Put in Maint Mode — only visible if device is not in maint mode. Disabled if user does not have maint mode privileges.

Remove — only visible if device is offline. Disabled if user does not have TSS configuration privileges.

Refresh — always visible; causes the GUI to refresh its cache by querying the TSS object in the server for its current configuration and status.

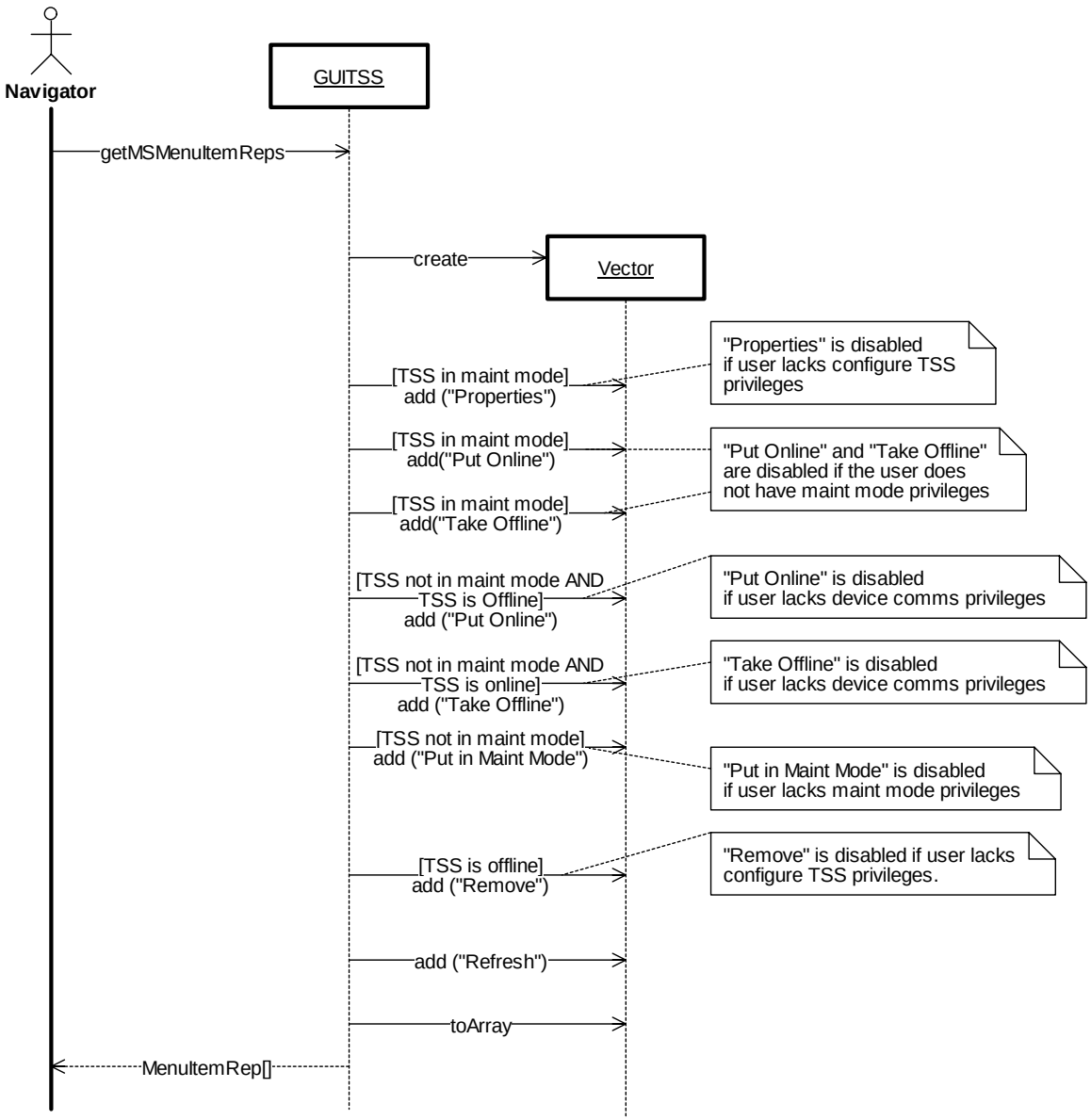

Figure 29. GUITSS:getMSMenuItemReps (Sequence Diagram)

3.3.2.11 GUITSS:getPropertyValue (Sequence Diagram)

When the navigator displays information for a TSS, it asks the GUITSS for the value for each property of the TSS to be displayed in a column of the navigator. The GUITSS returns property values based on its cached TSSConfiguration and TSSStatus objects.

Note that the GUITSSGroup tells the navigator all possible columns in its getAllNavProperties method. The navigator may have a filter applied, in which case the navigator will only ask the GUITSS for properties that are to be displayed.

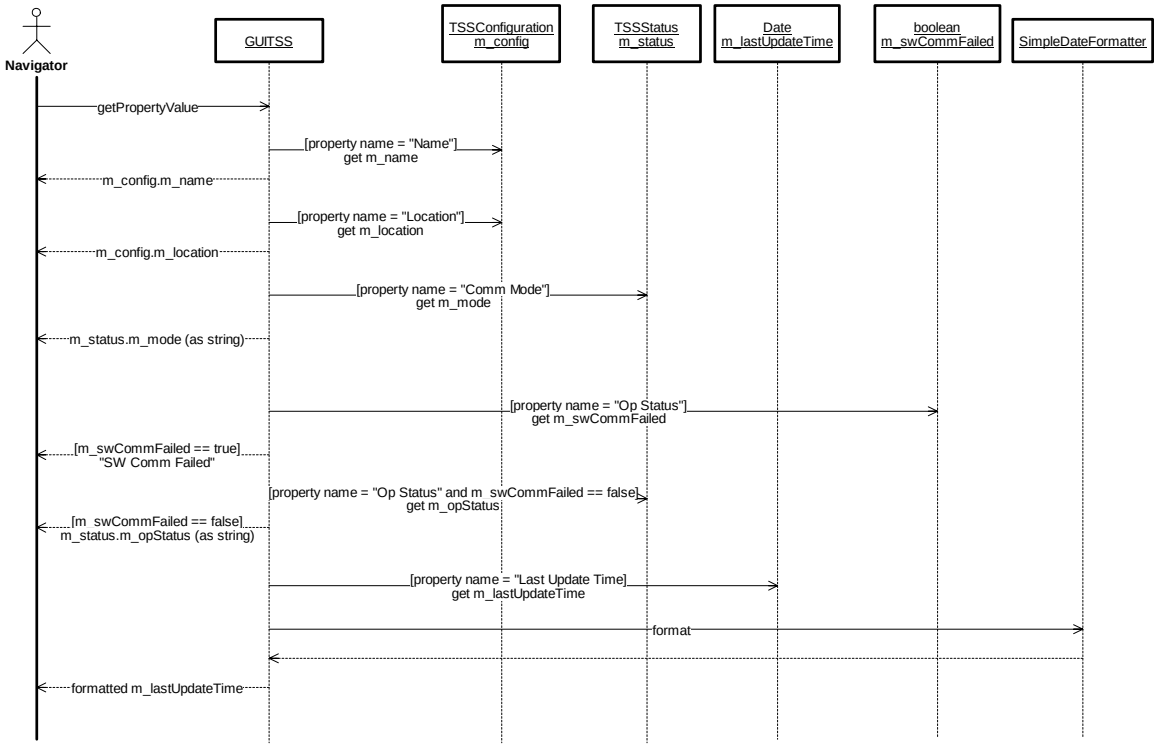

Figure 30. GUITSS:getPropertyValue (Sequence Diagram)

3.3.2.12 GUITSS:getSSMenuItemReps (Sequence Diagram)

When a user right clicks on a single TSS in the navigator, the navigator asks the GUITSS object for its list of menu items to appear in the context menu for the object. The possible menu items to appear in a TSS pop-up menu are as follows:

Properties — only visible if device is in maintenance mode. Disabled if user does not have TSS configuration privileges.

Put Online — only visible if device is not already online. Disabled if in maint mode and user does not have maint mode privileges. Disabled if offline and user does not have device comms privileges.

Take Offline — only visible if device is not already offline. Disabled if in maint mode and user does not have maint mode privelges. Disabled if online and user does not have device comms privileges.

Put in Maint Mode — only visible if device is not in maint mode. Disabled if user does not have maint mode privileges.

Remove — only visible if device is offline. Disabled if user does not have TSS configuration privileges.

Refresh — always visible; causes the GUI to refresh its cache by querying the TSS object in the server for its current configuration and status.

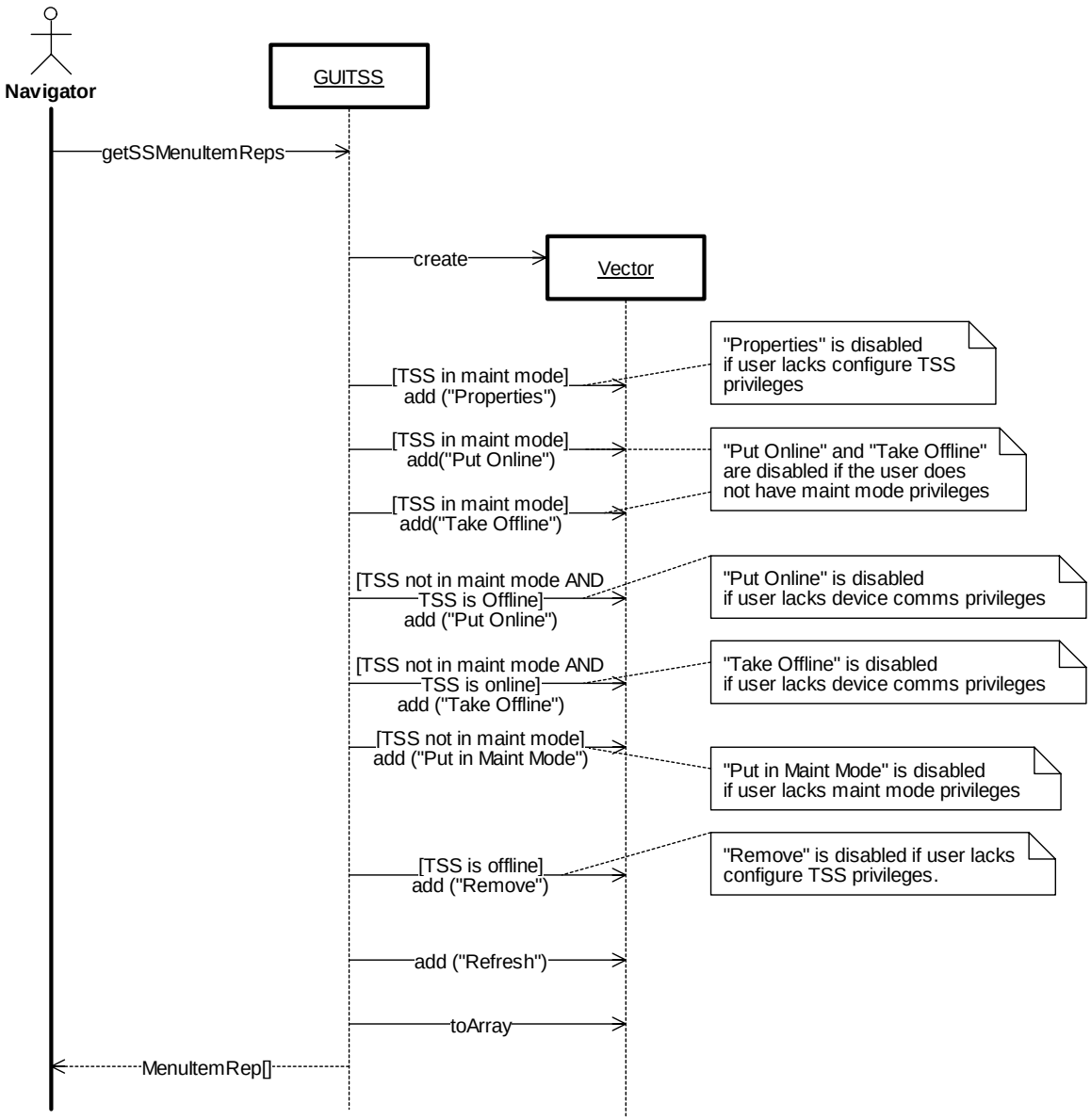

Figure 31. GUITSS:getSSMenuItemReps (Sequence Diagram)

3.3.2.13 GUITSS:putInMaintMode (Sequence Diagram)

When an administrator right clicks on a Transportation Sensor System that is not currently in maintenance mode, a menu is presented that contains the “Put In Maint Mode” menu item. When the user selects this item, the actionPerformed() method is called on the GUITSS object, which dispatches the processing to the putInMaintMode method. This command is processed by creating a CommandStatus object to allow the progress of the command to be tracked asynchronously and then passing this CommandStatus and the user’s token to the TransportationSensorSystem object’s putInMaintenanceMode() method.

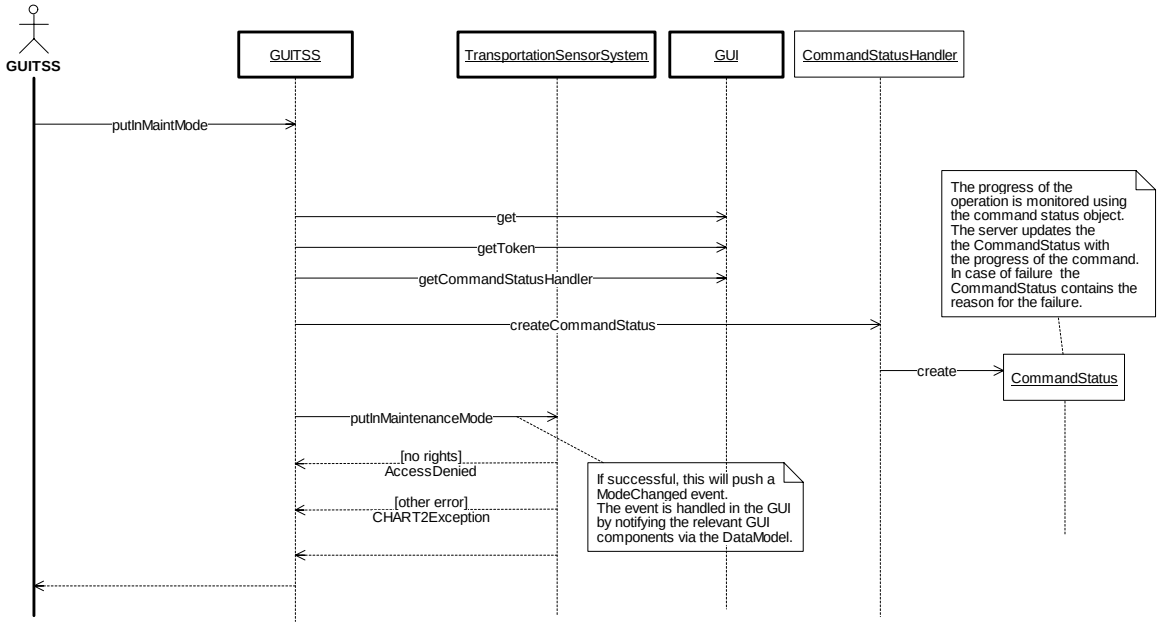

Figure 32. GUITSS:putInMaintMode (Sequence Diagram)

3.3.2.14 GUITSS:putOnline (Sequence Diagram)

When an administrator right clicks on a Transportation Sensor System that is not currently online, a menu is presented that contains the "Put Online" menu item. When the user selects this item, the actionPerformed() method is called on the GUITSS object, which dispatches the processing to the putOnline method. This command is processed by creating a CommandStatus object to allow the progress of the command to be tracked asynchronously and then passing this CommandStatus and the user's token to the TransportationSensorSystem object's putOnline() method.

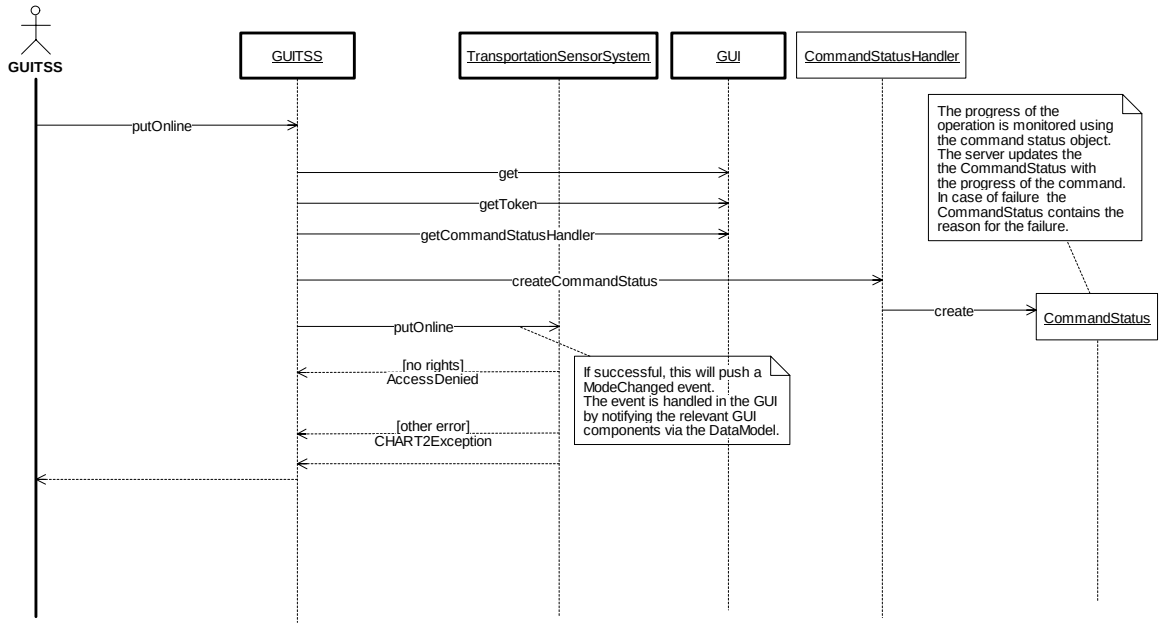

Figure 33. GUITSS:putOnline (Sequence Diagram)

3.3.2.15 GUITSS:refresh (Sequence Diagram)

When a user right clicks on a Transportation Sensor System, a menu is presented that contains the “Refresh” menu item. When the user selects this item, the actionPerformed() method is called on the GUITSS object, which dispatches the processing to the refresh method. This command is processed by calling the TSS object’s getConfiguration and getStatus methods, refreshing the GUITSS cache with the data that is obtained, and notifying the data model that the object has been updated.

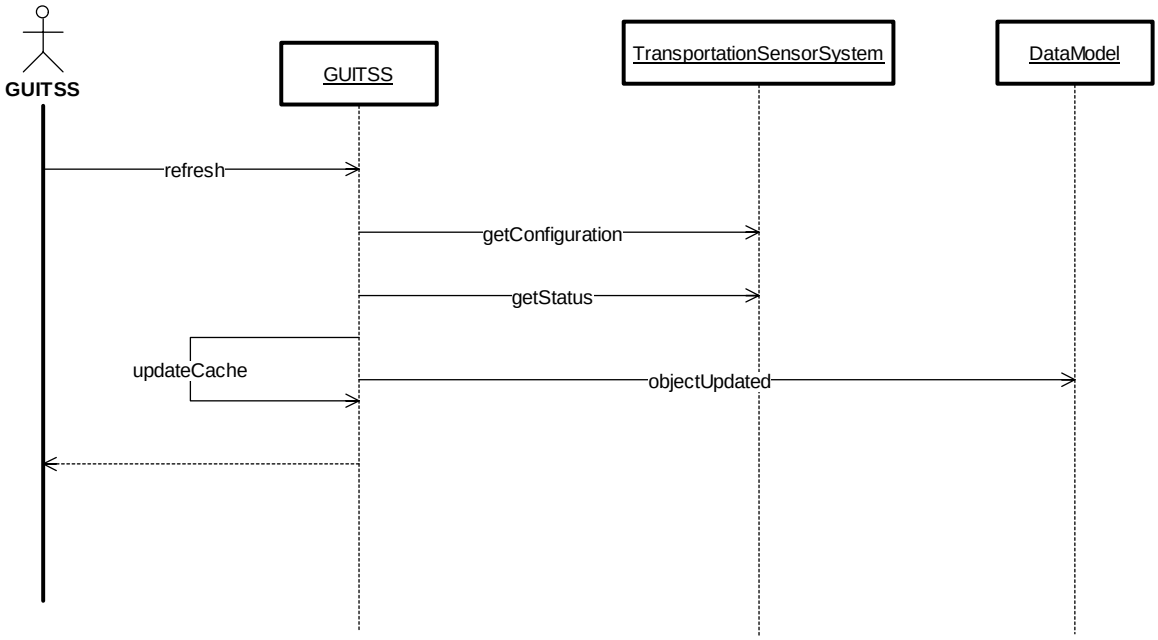

Figure 34. GUITSS:refresh (Sequence Diagram)

3.3.2.16 GUITSS:remove (Sequence Diagram)

When an administrator right clicks on a Transportation Sensor System that is offline, a menu is presented that contains the “Remove” menu item. When the user selects this item, the actionPerformed() method is called on the GUITSS object, which dispatches the processing to its remove method. The user is provided a warning to keep them from accidentally removing a TSS from the system. If the user chooses to continue with the removal, this command is processed by creating a CommandStatus object to allow the progress of the command to be shown in the command status window. Note that the server does not process this command asynchronously, so the command status is used only for consistency with other commands. The TransportationSensorSystem object’s remove() method is called to remove the object from the system.

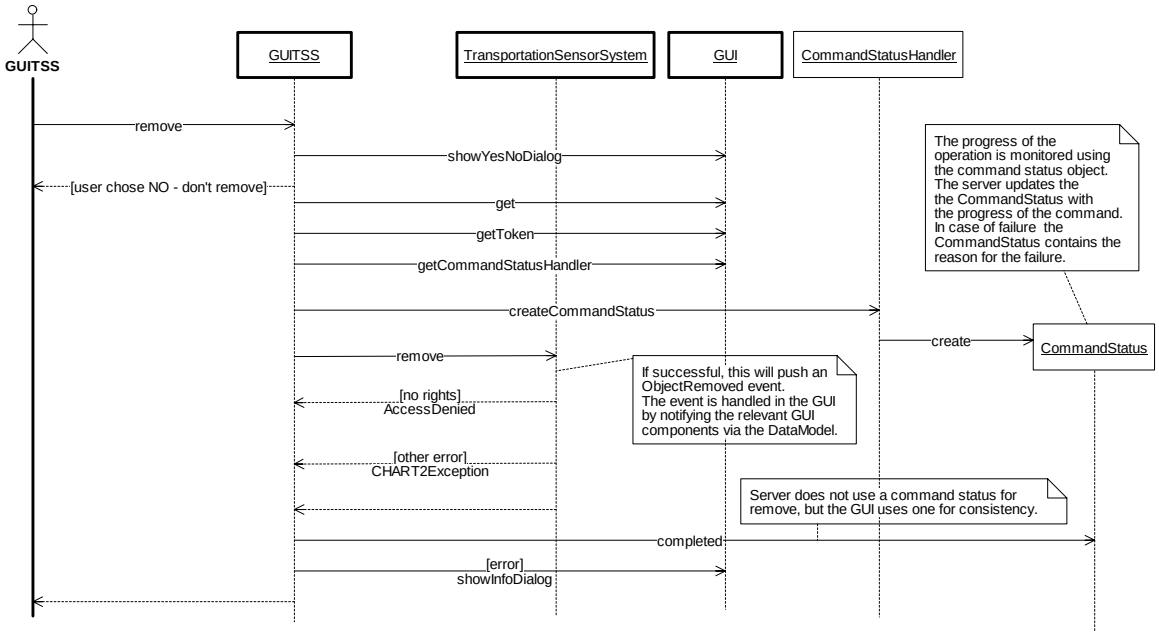

Figure 35. GUITSS:remove (Sequence Diagram)

3.3.2.17 GUITSS:takeOffline (Sequence Diagram)

When an administrator right clicks on a Transportation Sensor System that is not currently offline, a menu is presented that contains the "Take Offline" menu item. When the user selects this item, the actionPerformed() method is called on the GUITSS object, which dispatches the processing to its takeOffline method. This command is processed by creating a CommandStatus object to allow the progress of the command to be tracked asynchronously and then passing this CommandStatus and the user's token to the TransportationSensorSystem object's takeOffline() method.

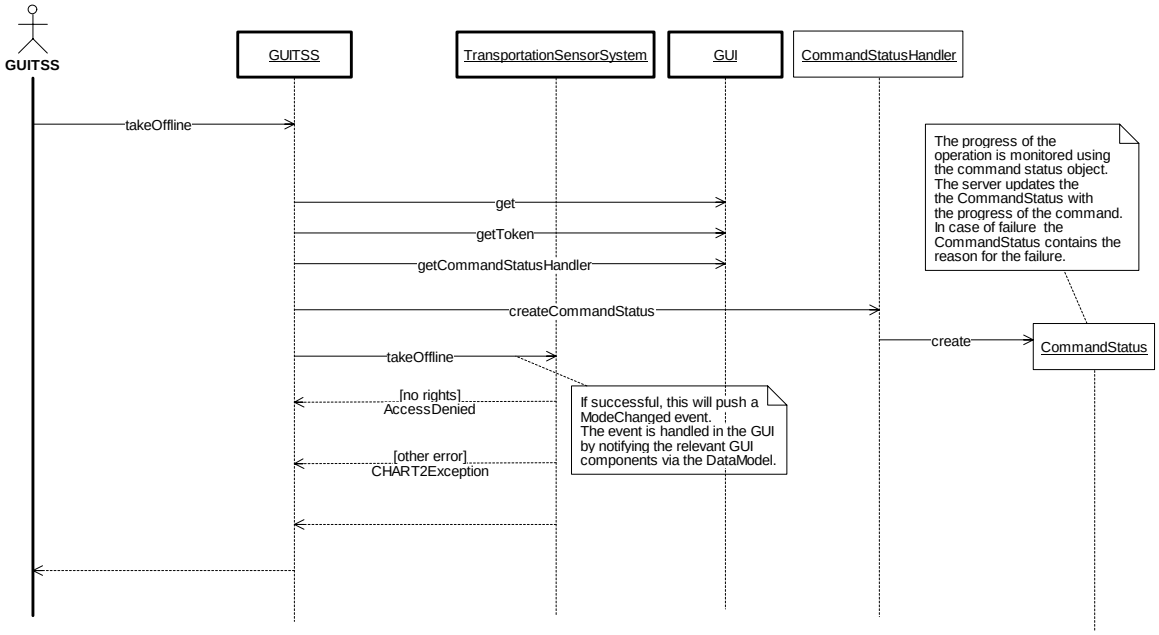

Figure 36. GUITSS:takeOffline (Sequence Diagram)

3.3.2.18 GUITSSGroup:actionPerformed (Sequence Diagram)

When the user selects a menu item from the pop-up created by right clicking on the TSS folder in the navigator, the GUITSSGroup object's actionPerformed method is called with information about which menu item was selected. The GUITSSGroup gets the list of model supporters from the GUITSSModule and asks each if it can create the type of TSS indicated by the menu item that was selected. When a model supporter is found that can create a GUITSS object, the search is over. The GUITSSGroup then calls the doProperties method on the GUITSS object that was created, allowing the user to enter configuration data. See the GUIRTMS:doProperties sequence for details.

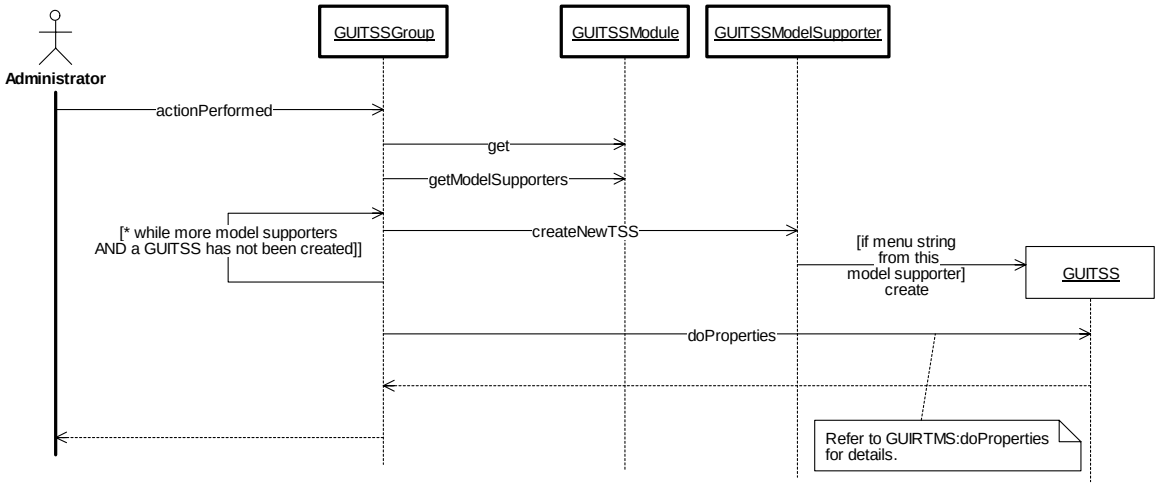

Figure 37. GUITSSGroup:actionPerformed (Sequence Diagram)

3.3.2.19 GUITSSGroup:getAllNavProperties (Sequence Diagram)

When a NavClassFilter such as the GUITSSGroup is selected by the user, the navigator needs to know what columns to show in the navigator. It determines the columns by calling the filter's getAllNavProperties() method. The GUITSSGroup implements this method by returning an array of the following columns:

Location — Text description as entered by the administrator

Communication Mode — Online, Offline, Maint Mode

Operational Status — OK, Hardware Failed, Comm Failed, SW Comm Failed

Update Time — Last time the cached info for this TSS was updated.

Note that these are in addition to the standard navigator columns that contain an icon and the name.

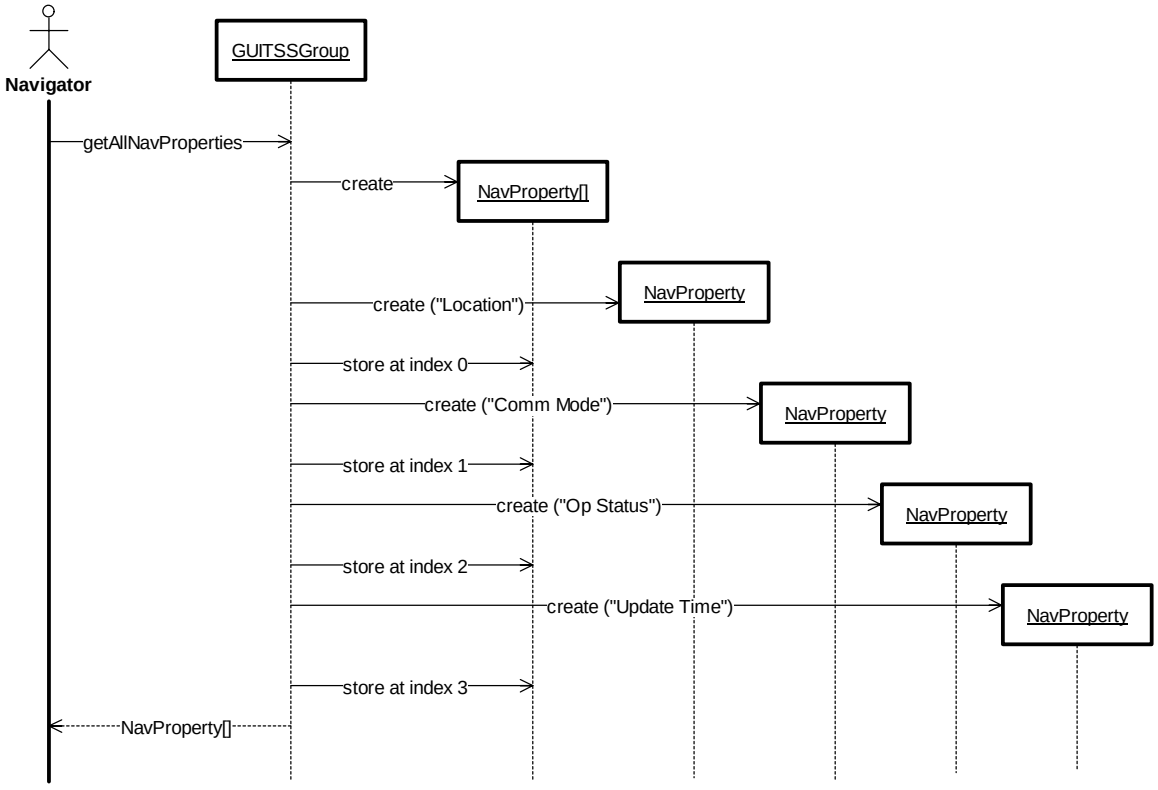

Figure 38. GUITSSGroup:getAllNavProperties (Sequence Diagram)

3.3.2.20 GUITSSGroup:getSSMenuItemReps (Sequence Diagram)

When the user right clicks on the GUITSSGroup in the navigator tree, the navigator calls the GetMenuItemReps() method to allow the GUITSSGroup to add items to the pop-up menu. The GUITSSGroup asks all model supporters in the GUITSSModule to supply their own list of items for the pop-up menu, and all are returned as a single array of MenuItemRep objects.

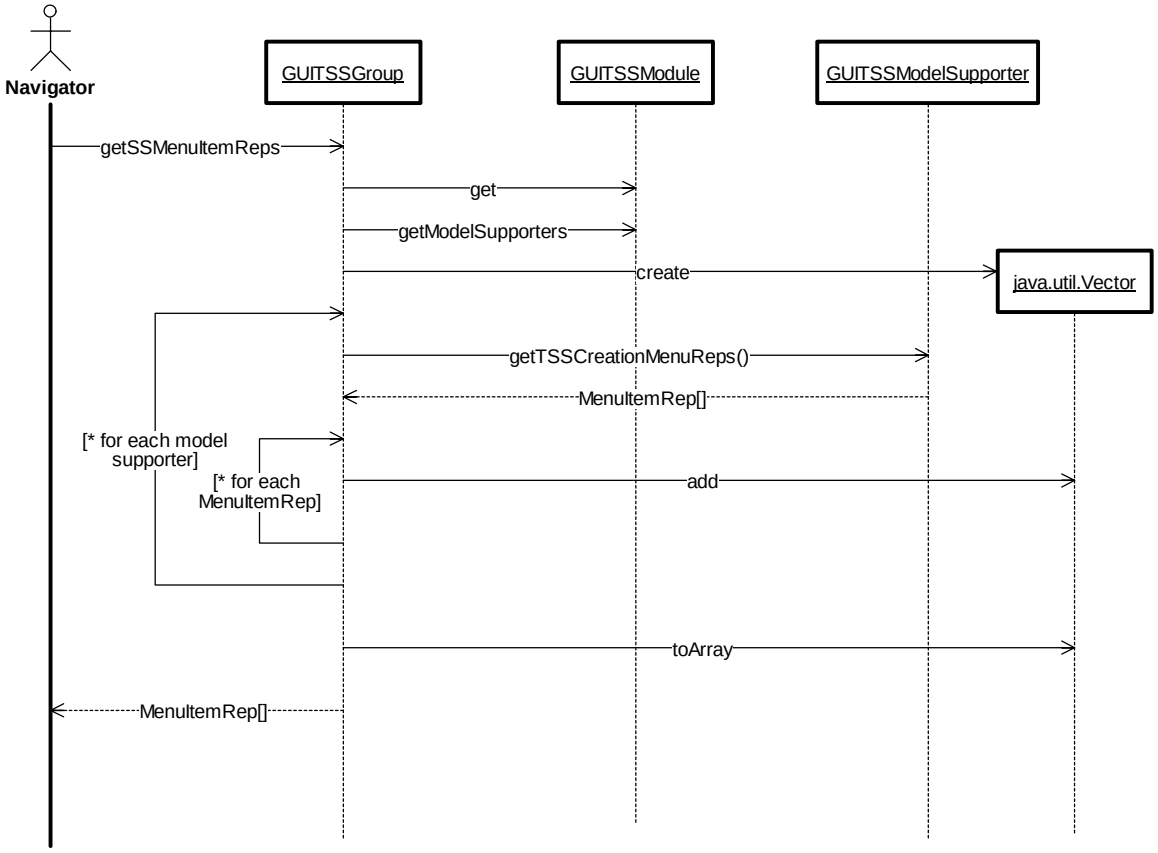

Figure 39. GUITSSGroup:getSSMenuItemReps (Sequence Diagram)

3.3.2.21 GUITSSModule:discoverEventChannels (Sequence Diagram)

At startup and periodically during the life of the GUI, the GUI calls each module to allow it to discover CORBA event channels that are of interest to the module. When the GUITSSModule's discoverEventChannels() method is called, it retrieves its CORBA object reference from the POA and then calls a GUI utility method that discovers all CORBA event channels of the specified name and connects the module as a PushConsumer to the event channel.

Note: The TSSManagementModule in the server uses two event channels, one for pushing status and configuration changes, and another to push data collected from sensors. At this time, this module only connects to the status channel because the sensor data is not displayed in the GUI.

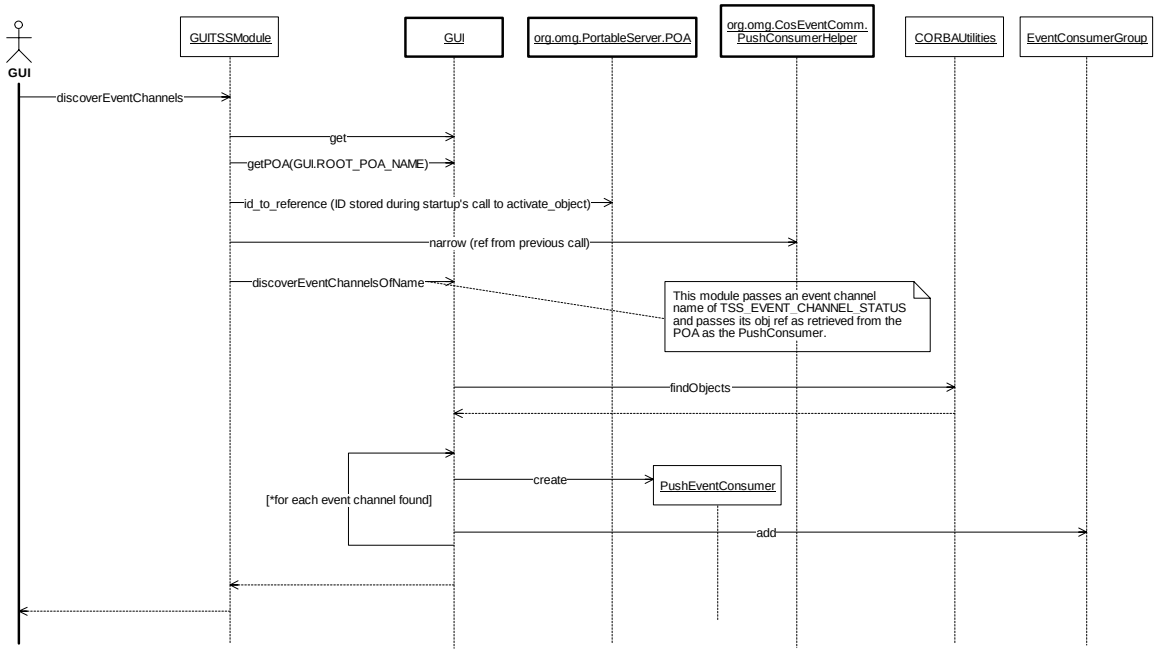

Figure 40. GUITSSModule:discoverEventChannels (Sequence Diagram)

3.3.2.22 GUITSSModule:discoverObjects (Sequence Diagram)

At startup and periodically during the life of the GUI application, the GUI object calls each module to have them discover distributed objects for which the module provides GUI access. Upon receiving this call, the GUITSSModule makes a trader query (via the CorbaUtilities class) to discover all TransportationSensorSystemFactory objects in the system. The module then gets a list of TransportationSensorSystem objects from each factory, and for each TransportationSensorSystem checks to see if the GUI is already aware of the object by checking in the DataModel. If the object is not already known to the GUI, the module asks each of its model supporters to create a wrapper for the object. The model supporter that supports the specific subclass of TransporationSensorSystem creates a derived GUITSS object to wrap the CORBA object, while model supporters that don't support the subclass return null. After a model supporter is found that can create a wrapper, the wrapper object is added to the DataModel.

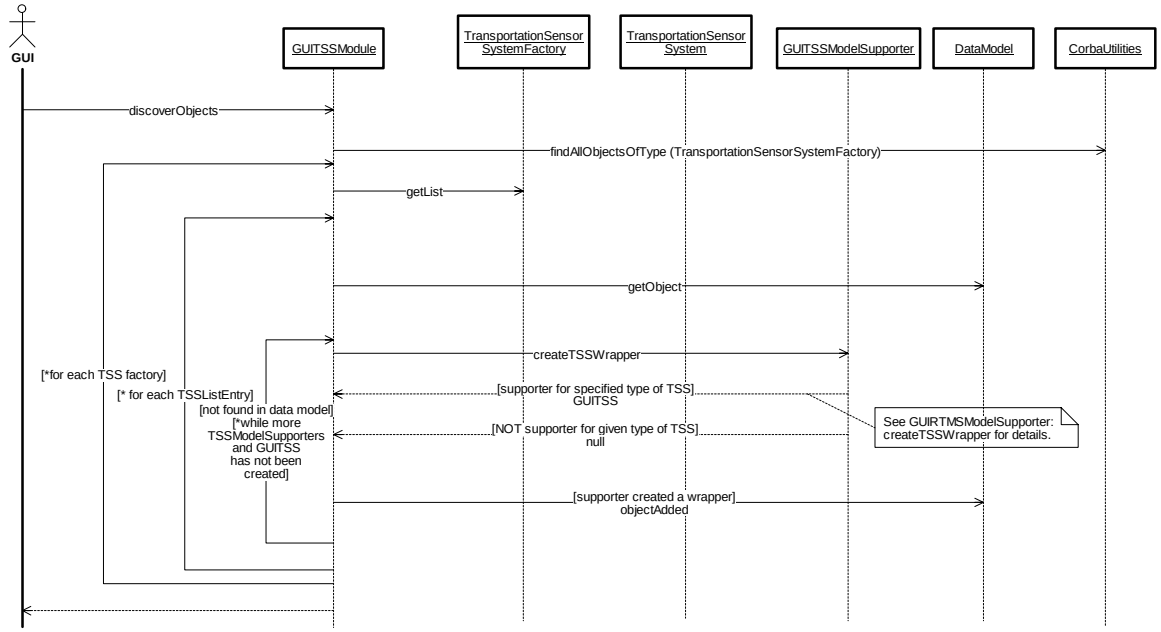

Figure 41. GUITSSModule:discoverObjects (Sequence Diagram)

3.3.2.23 GUITSSModule:getMenuItemReps (Sequence Diagram)

When the user right clicks in the open area of the task bar on the GUI, the GUI calls each installed module to allow them to add menu items to the general GUI pop-up menu. The GUITSSModule has no items to contribute, so it returns a zero length array.

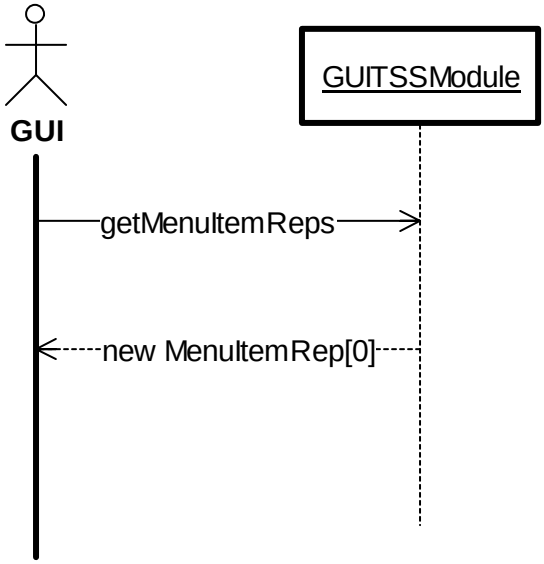

Figure 42. GUITSSModule:getMenuItemReps (Sequence Diagram)

3.3.2.24 GUITSSModule:handleCommand (Sequence Diagram)

When the user selects a menu item from the GUI's general pop-up menu (as constructed by calling each module's getMenuItemReps() method), the GUI passes the event to each module to allow them to handle it. Because the GUITSSModule contributes no items to the general GUI pop-up, it does no processing in its handleCommand() method.

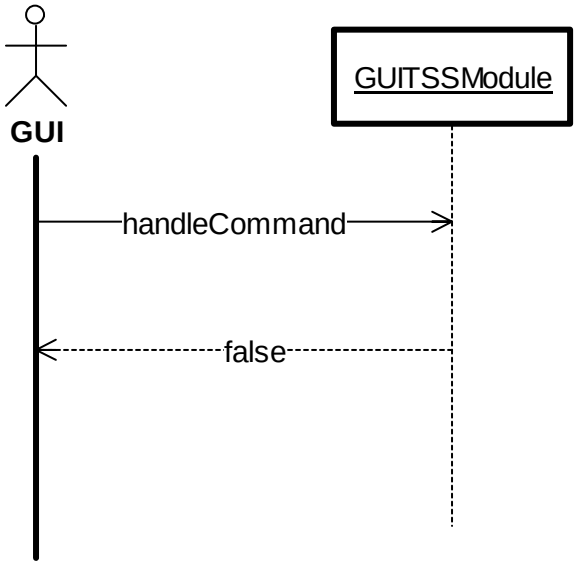

Figure 43. GUITSSModule:handleCommand (Sequence Diagram)

3.3.2.25 GUITSSModule:loggedIn (Sequence Diagram)

The GUI notifies the GUITSSModule when a user logs into the GUI. The GUITSSModule has no processing to perform and just returns.

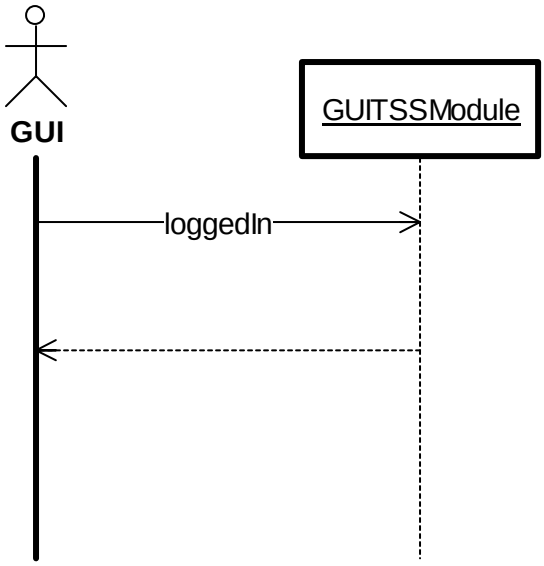

Figure 44. GUITSSModule:loggedIn (Sequence Diagram)

3.3.2.26 GUITSSModule:push (Sequence Diagram)

The GUITSSModule is a PushConsumer and receives all events that the server pushes on the TSS status event channel. When an event is received, the module creates a GUITSSEventHandler object and passes it to the java swing engine so it can be invoked on the main GUI thread. When the GUITSSEventHandler is run from the GUI thread, it calls back into the GUITSSModule handleCORBAEvent method to process the event data. The processing of the event data involves determining the discriminator contained in the TSSEvent object and processing based on the type of event.

For ConfigChanged, OpStatusChanged, and ModeChanged events, the GUITSS object for which the event applies is retrieved from the DataModel and its cached values are updated, causing the GUITSS to notify the DataModel that it has been updated.

When an ObjectAdded event is received, the DataModel is checked to make sure the object does not already exist. If the object does not exist, the GUITSSModule asks each of the installed model supporters to create a wrapper for the object that has been added. The model supporter for the type of TSS that was added creates a GUITSS derived object, and the object is passed to the DataModel in an ObjectAdded call.

When an ObjectRemoved event is received, the ObjectRemoved method is called on the DataModel to remove the object from the GUI.

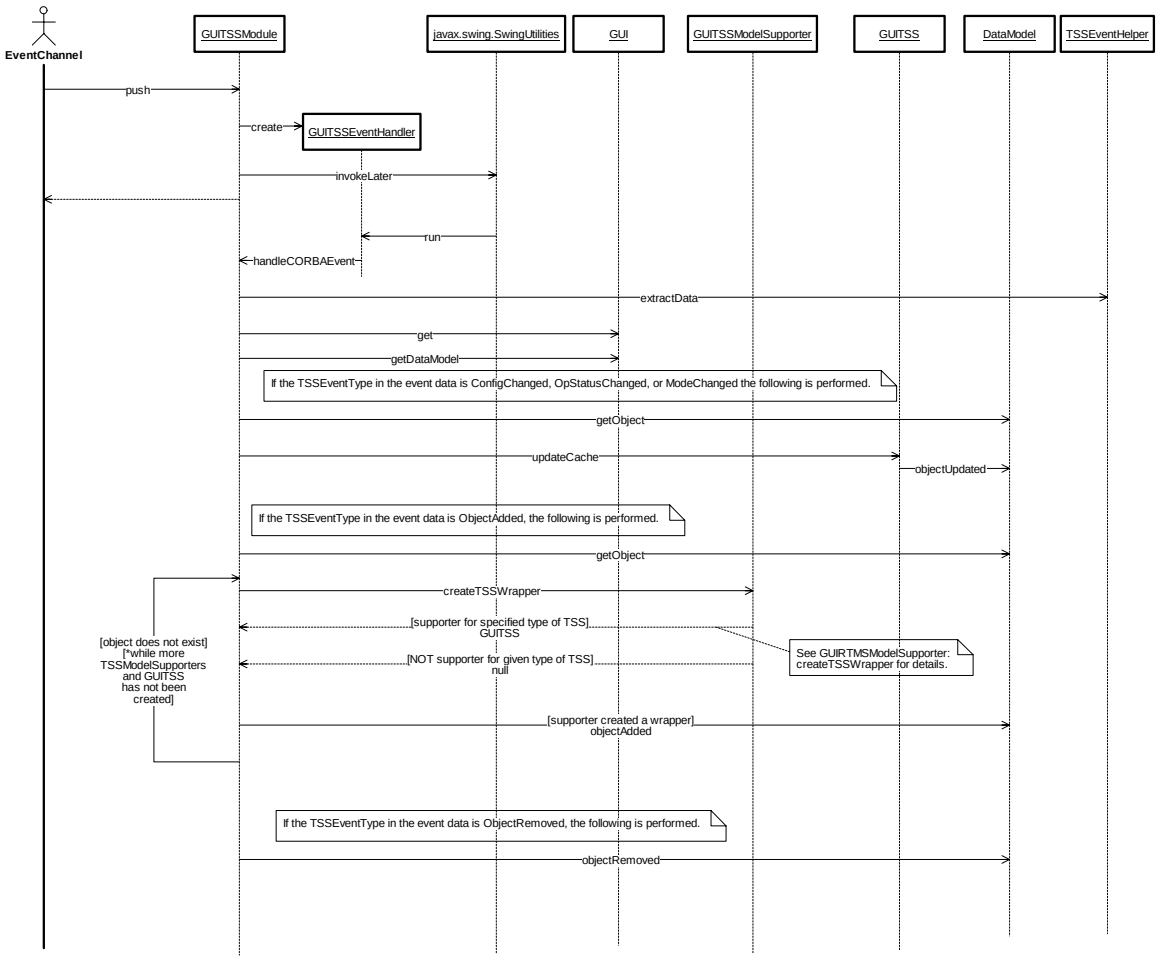

Figure 45. GUITSSModule:push (Sequence Diagram)

3.3.2.27 GUITSSModule:loggedOut (Sequence Diagram)

The GUI notifies the GUITSSModule when the user logs out from the GUI. The GUITSSModule has no processing to perform in response to this notification and simply returns.

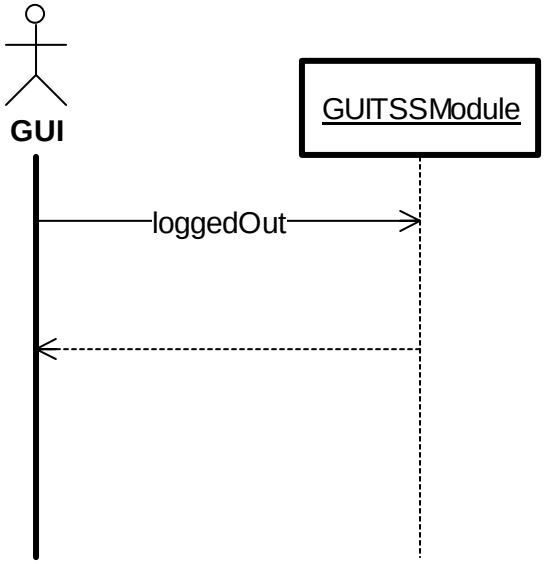

Figure 46. GUITSSModule:loggedOut (Sequence Diagram)

3.3.2.28 GUITSSModule:shutdown (Sequence Diagram)

When the GUI instructs the GUITSSModule to shutdown, the GUITSSModule disconnects itself from the ORB and releases its reference to each GUITSSModelSupporter it constructed during startup, allowing these classes to be garbage collected.

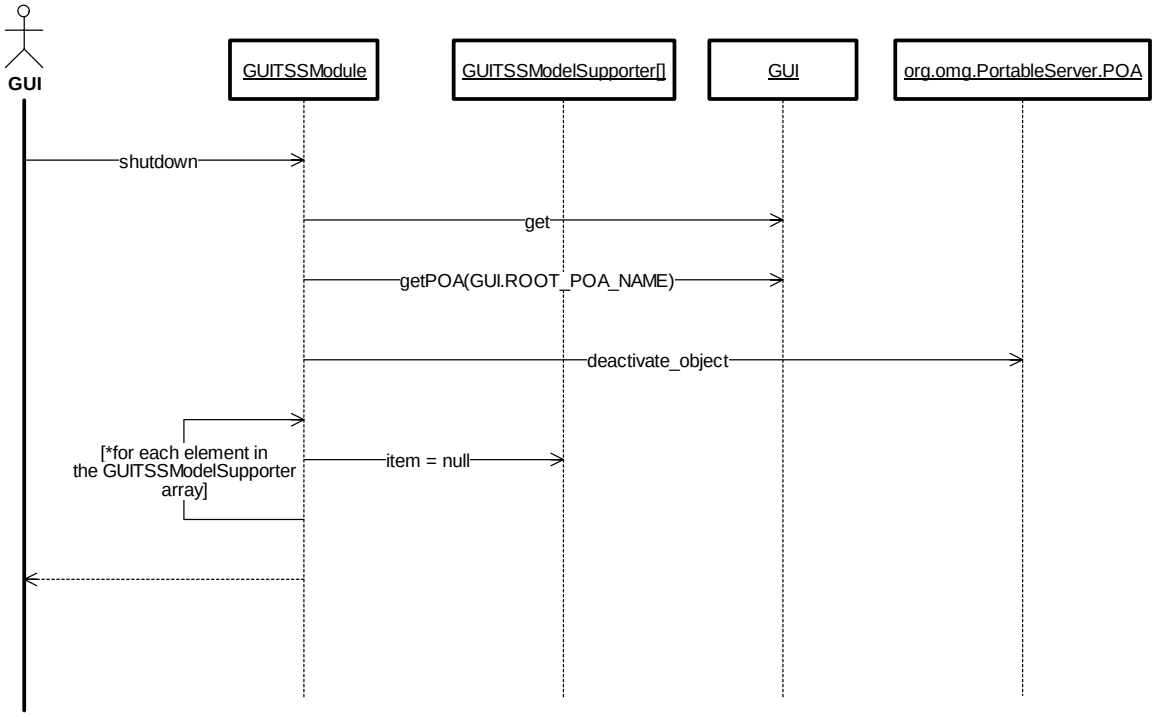

Figure 47. GUITSSModule:shutdown (Sequence Diagram)

3.3.2.29 GUITSSModule:startup (Sequence Diagram)

When the GUITSSModule is started by the GUI, it constructs model supporters for the various types of TSS models it supports. At the time of this writing, the only model supported is RTMS. The module also connects itself to the ORB so it can receive CORBA calls via its PushConsumer interface.

Finally, the module initializes filters that it owns. It first asks the GUI's filter manager for all filters previously created, either as a user preference or as a global addition by an administrator. This module activates each of these pre-existing filters, making them available in the navigator. If previously created filters do not exist, this module must create its Group (TSS) for the navigator tree and activate it, making it available in the navigator.

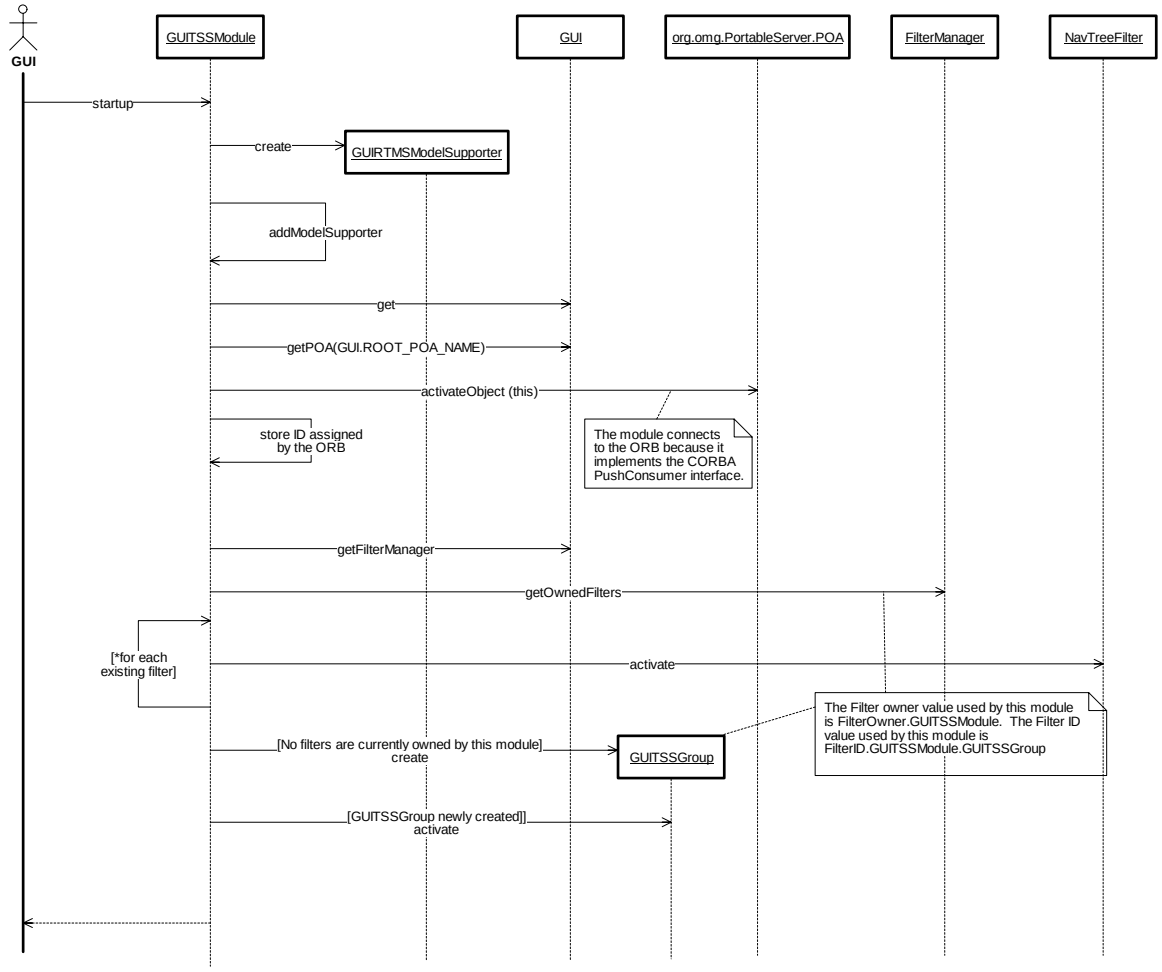

Figure 48. GUITSSModule:startup (Sequence Diagram)

3.4 DeviceUtility

3.4.1 PortLocatorClasses (Class Diagram)

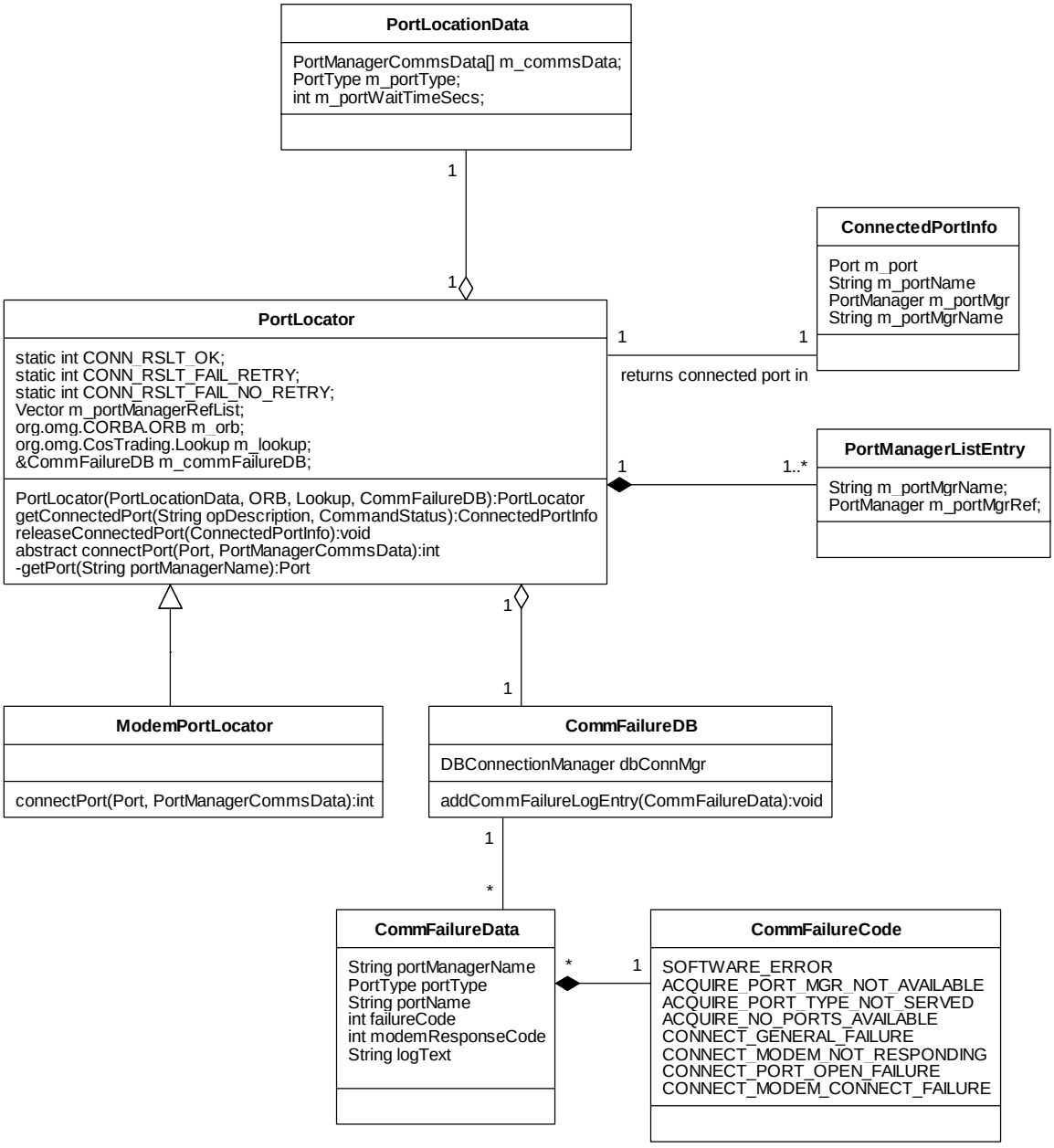

Figure 49. PortLocatorClasses (Class Diagram)

3.4.1.1 CommFailureCode (Class)

This class defines static values to be used to specify the type of comm failure in a CommFailureData object.

3.4.1.2 CommFailureData (Class)

This class holds data to be passed to the CommFailureDB class to be logged in the Comm failure log in the database.

3.4.1.3 CommFailureDB (Class)

This class is a utility used to log an entry in the Comm Failure log table in the database. This table is used to log details about any comm failure that occurs in the system.

3.4.1.4 ConnectedPortInfo (Class)

This class holds data pertaining to a port that was acquired and connected via the PortLocator.

typedef

3.4.1.5 ModemPortLocator (Class)

This class provides an implementation of the PortLocator's abstract connectPort() method that can connect a ModemPort that has been acquired by the PortLocator base class. This derived class logs information in the comm failure database table relating to connection problems that may occur.

3.4.1.6 PortLocationData (Class)

This class contains configuration data that specifies the communication server(s) to use to communicate with a device.

m_commsData — One or more objects identifying the communications server (PortManager) to use to communicate with the device, in order of preference.

m_portType — The type of port to use to communicate with the device (ISDN modem, POTS modem, direct, etc.)

m_portWaitTimeSecs — The maximum number of seconds to wait when attempting to acquire a port from a port manager.

3.4.1.7 PortLocator (Class)

The PortLocator is a utility class that helps one to utilize the fault tolerance provided by the deployment of many PortManagers. The PortLocator is initialized by specifying a preferred PortManager and optionally one or more alternate PortManagers using a PortLocationData object.

When asked to get a connected port, the PortLocator first attempts to acquire a port from the preferred PortManager and then calls its abstract connectPort() method (implemented by derived classes) to attempt to connect to the port. If a failure occurs, the PortLocator retries the sequence using the next PortManager in the list. The list may contain the same port manager multiple times to have retries occur on the same port manager prior to moving to another. In the event that the PortLocator will perform a retry on the same port manager, it holds the previously acquired port while performing the retry to avoid having the port manager return the same port during the retry. When a different port is acquired during a retry on the same port manager, the port is released (prior to connecting the 2nd port).

3.4.1.8 PortManagerListEntry (Class)

This class is used by the PortLocator to map object identifiers to object references for PortManager objects.

3.4.2 Sequence Diagrams

3.4.2.1 ModemPortLocator:connectPort (Sequence Diagram)

This sequence shows the ModemPortLocator processing involved when its base class invokes the virtual connectPort method. The ModemPortLocator casts the port retrieved by the base class into a ModemPort and calls its connect method. The ModemPortLocator then interprets the results of the connect call and returns a code to the base class to indicate *success, failure - retries should not be attempted,* or *failure - retries may be attempted.* The *failure - retries should not be attempted* result code is used in situations where a retry on a different port would likely yield the same result or when a software failure is encountered. If a failure occurs, detailed comm failure data is logged to the database comm failure table.

Note: this feature is controlled by the constructor of the class, allowing test programs to use this class without requiring a database connection.

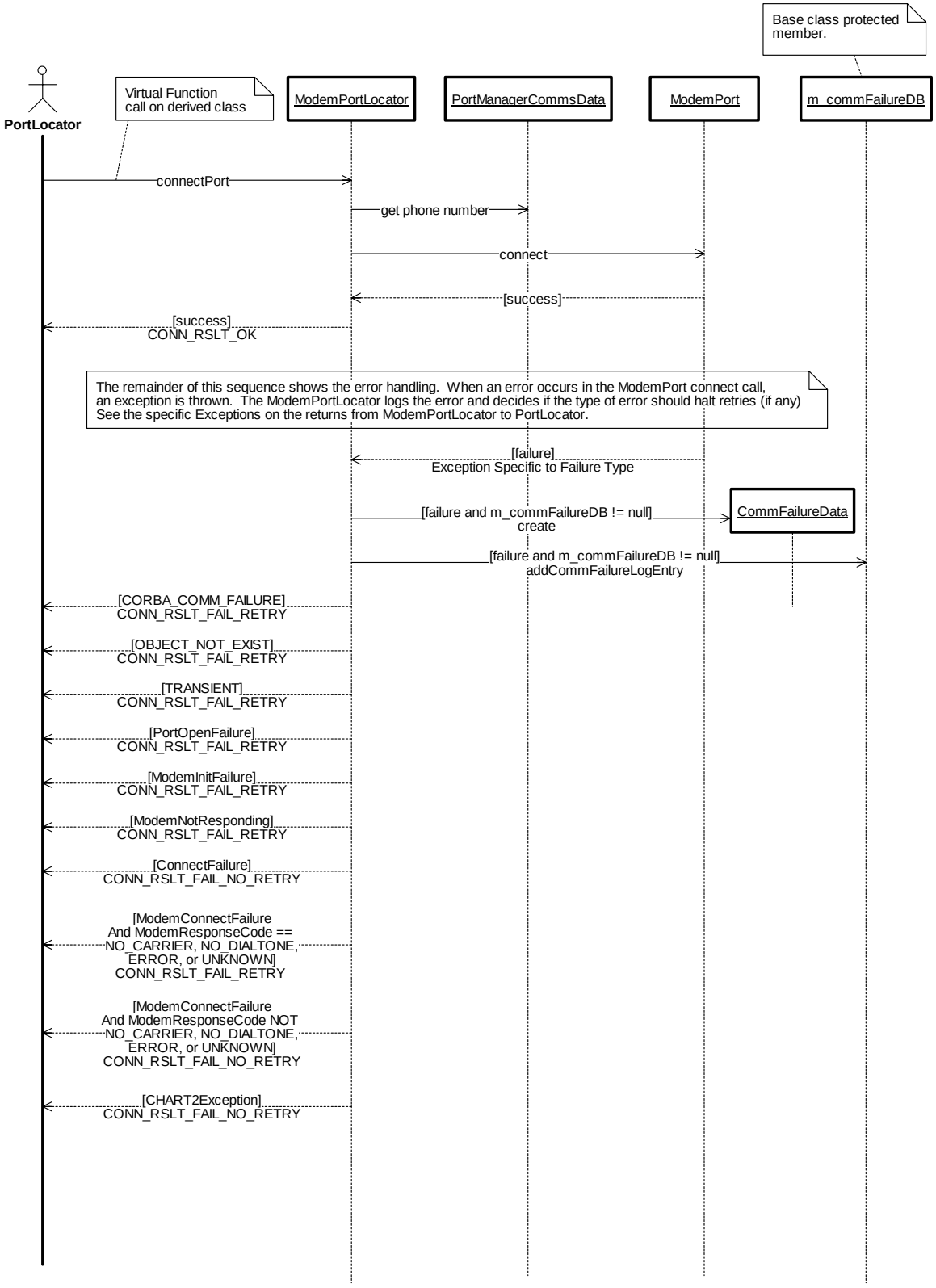

Figure 50. ModemPortLocator:connectPort (Sequence Diagram)

3.4.2.2 PortLocator:getConnectedPort (Sequence Diagram)

The getConnectedPort method of the PortLocator utility uses the list of PortManager names and associated connection information (such as phone number to use) to attempt to acquire a port and connect it to the remote destination. Retry logic exists to try each PortManager in succession until a port is successfully connected or an attempt to connect fails and the type of failure is not likely to benefit from a retry on a different port. The list of port manager names can contain duplicate entries to cause the port locator to use a different port on the same port manager. When this is the case, the port locator must hold the previously acquired port while it attempts to get an additional port from the port manager to ensure the port manager doesn't return the same port twice.

The connection logic is carried out in the derived class connectPort() method, for this logic varies depending on the type of port requested. A private getPort() method handles logic to retrieve a port from a single port manager and process errors. Sequences for these methods exist in the ModemPortLocator:connectPort() sequence and the PortLocator:getPort() sequence.

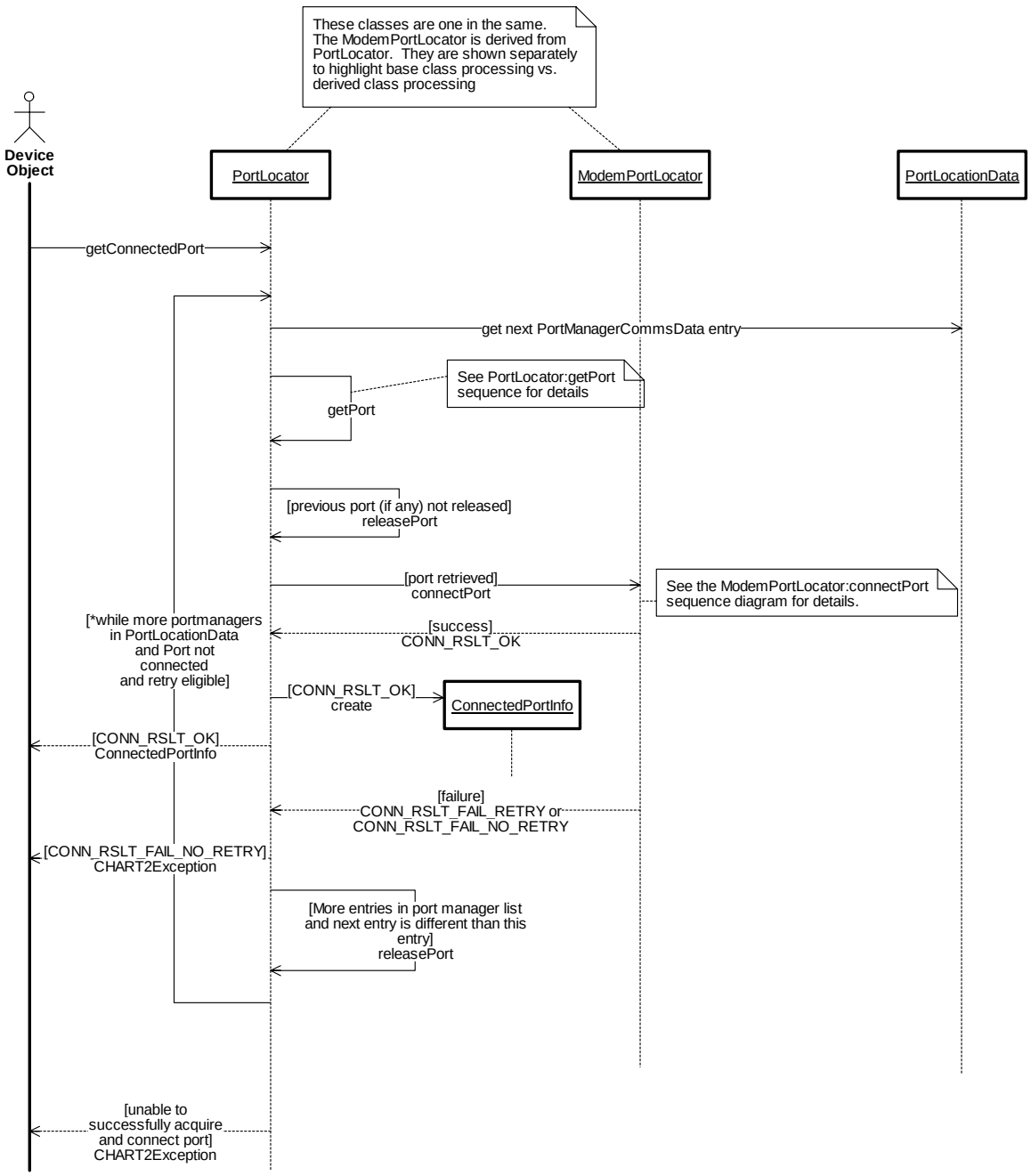

Figure 51 PortLocator:getConnectedPort (Sequence Diagram)

3.5 CHARTWebModule

3.5.1 CHARTWebModuleClasses (Class Diagram)

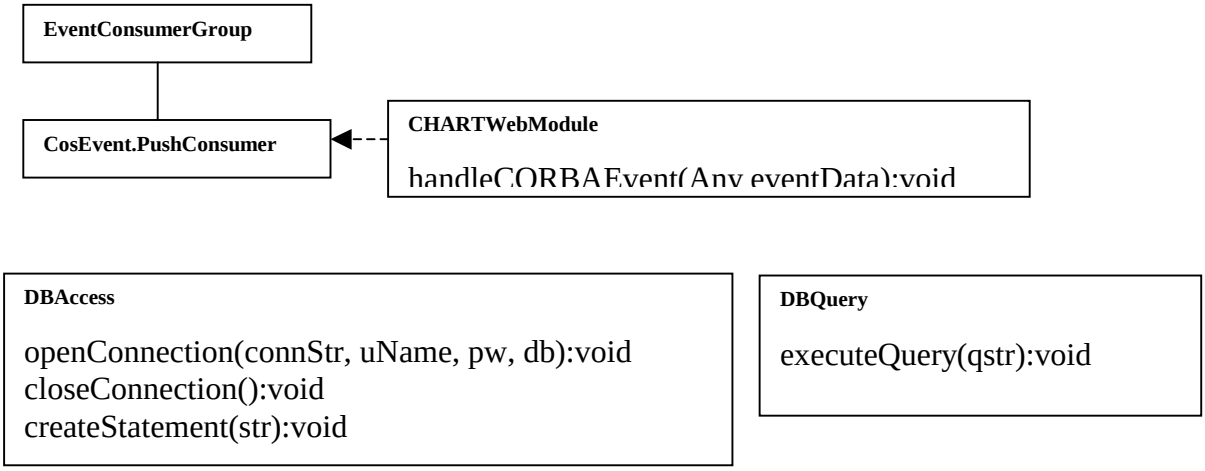

Figure 52 CHARTWebModuleClasses (Class Diagram)

3.5.1.1 CosEvent.PushConsumer (Class)

The PushConsumer interface is the interface to an event channel that a supplier of information uses to push event updates to consumers who have previously attached to the channel.

3.5.1.2 EventConsumerGroup (Class)

This class represents a collection of event consumers that will be monitored to verify that they do not lose their connection to the CORBA event service. The class will periodically ask each consumer to verify its connection to the event channel on which it is dependent to receive events.

3.5.1.3 CHARTWebModule (Class)

This class implements the CORBA PushConsumer interface and is therefore a CORBA object that is connected to the ORB and is called remotely by an EventChannel (via its PushConsumer push() method) when data is pushed on the channel. This module connects to the TSS status and event channels that exist in the system to allow this module to be notified of status and configuration changes to TSS objects.

3.5.1.4 DBAccess (Class)

This class creates a database connection and stores connection information for reconnecting if a disconnect occurs.

3.5.1.5 DBQuery (Class)

This class allows a database command to be sent to the CHARTWeb database through an established connection.

3.5.2 CHARTWebModule:ModeChanged (Sequence Diagram)

The CHART II system pushes an event to the Chartweb.RTMSClient when the mode (online, offline, or maintenance) for an RTMS is changed. The Chartweb.RTMSClient handles this event by updating all rows in the web database for the given RTMS ID with the new mode. The web map will only show data for rows in the database that are marked “online.”

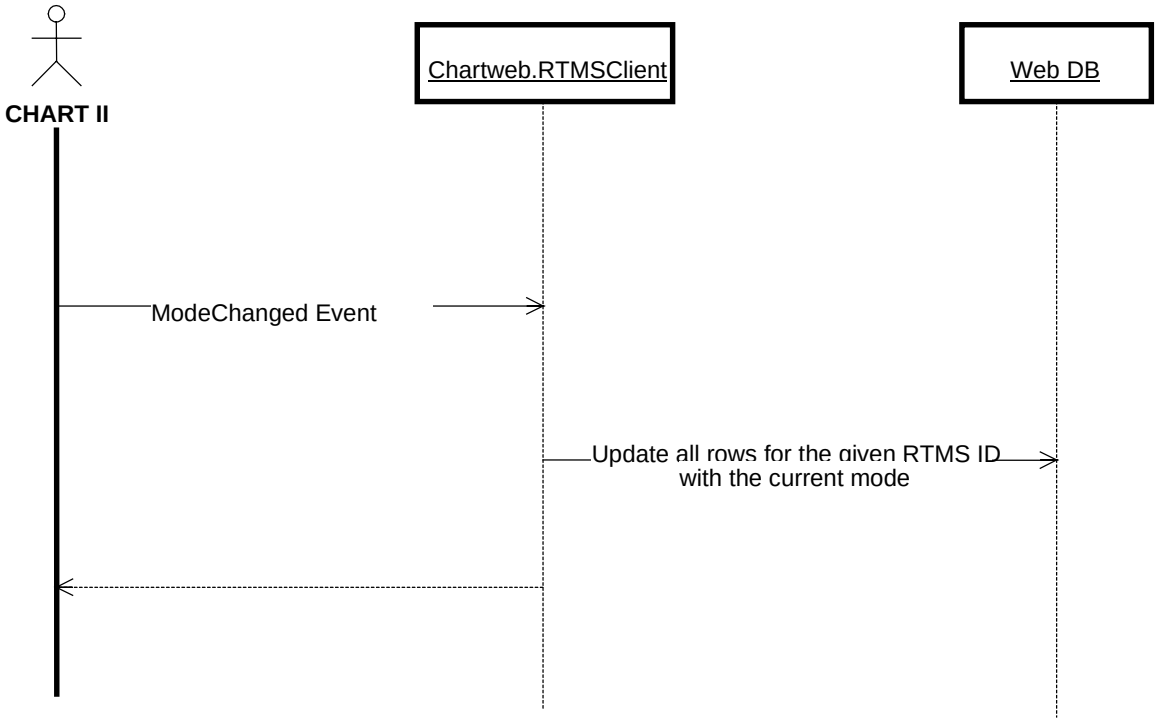

Figure 53. CHARTWebModule:ModeChanged (Sequence Diagram)

3.5.3 CHARTWebModule:OpStatusChanged (Sequence Diagram)

The CHART II system pushes an event to the Chartweb.RTMSClient when the status (OK, Communications Failure, or Hardware Failure) for an RTMS is changed. The Chartweb.RTMSClient handles this event by updating all rows in the web database for the given RTMS ID with the new status.

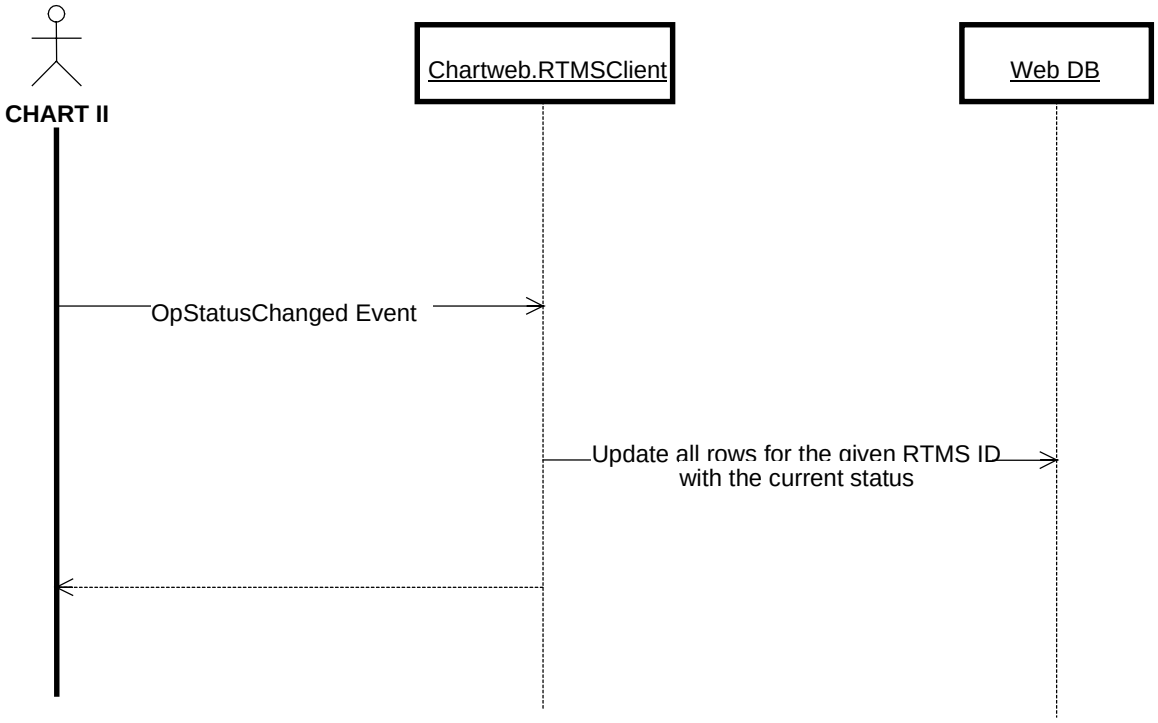

Figure 54. CHARTWebModule:OpStatusChanged (Sequence Diagram)

3.5.4 CHARTWebModule:currentStatusPush (Sequence Diagram)

When updates to the status of RTMS devices are received on the Data event channel, the Chartweb.RTMSClient makes appropriate updates to the web database.

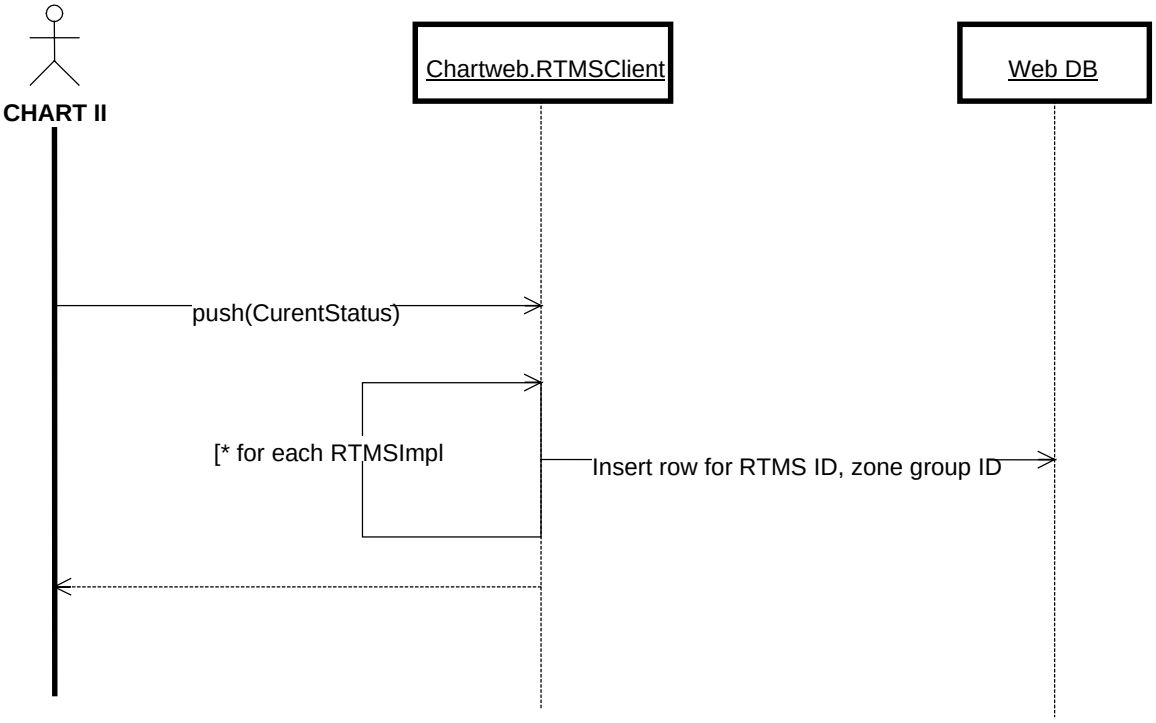

Figure 55. CHARTWebModule:currentStatusPush (Sequence Diagram)

3.5.5 CHARTWebModule:ConfigChanged (Sequence Diagram)

When a push(ConfigChanged) message is received on the Status event channel, the Chartweb.RTMSClient makes no updates to the web database.

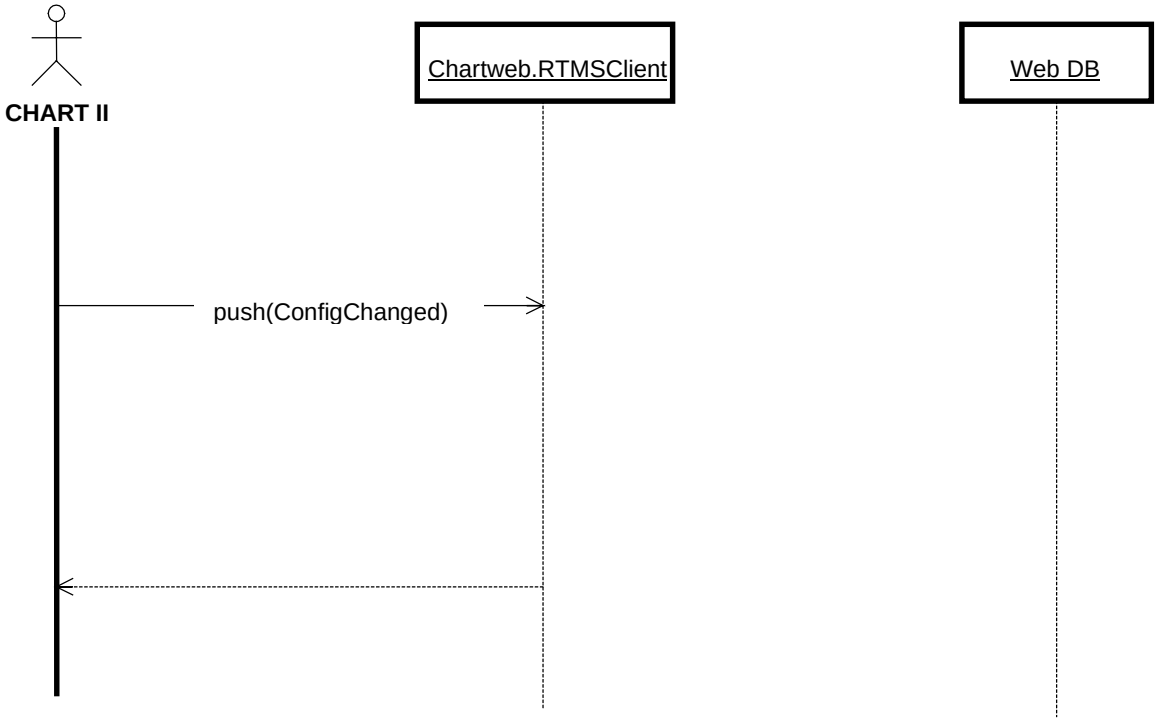

Figure 56. CHARTWebModule:ConfigChanged (Sequence Diagram)

3.5.6 CHARTWebModule:Initialize (Sequence Diagram)

When the Chartweb.RTMSClient is first initialized, it prepares itself to receive asynchronous updates of RTMS status from the CHART II system and then gets the current state of RTMS objects from CHART II and sets the web database data to match the CHART II current state. After this initialization is complete, updates to the status of RTMS devices are received asynchrounously as changes occur, at which time the Chartweb.RTMSClient makes appropriate updates to the web database. See the other CHARTWebModule sequence diagrams for details.

If the Chartweb.RTMSClient is unable to contact the CHART II trader or event service, all devices in the web DB are marked offline and this sequence is retried periodically.

If individual RTMS objects within the CHART II system cannot be contacted, rows for the specific RTMS are marked offline in the web database and this sequence is retried periodically.

This sequence is also carried out when the Chartweb.RTMSClient suspects the CHART II system has gone down due to the lack of events received from CHART II. When this occurs, this initialization sequence will serve to verify the current status of the RTMS objects or to confirm that CHART II is not fully available and to mark the appropriate RTMS objects offline.

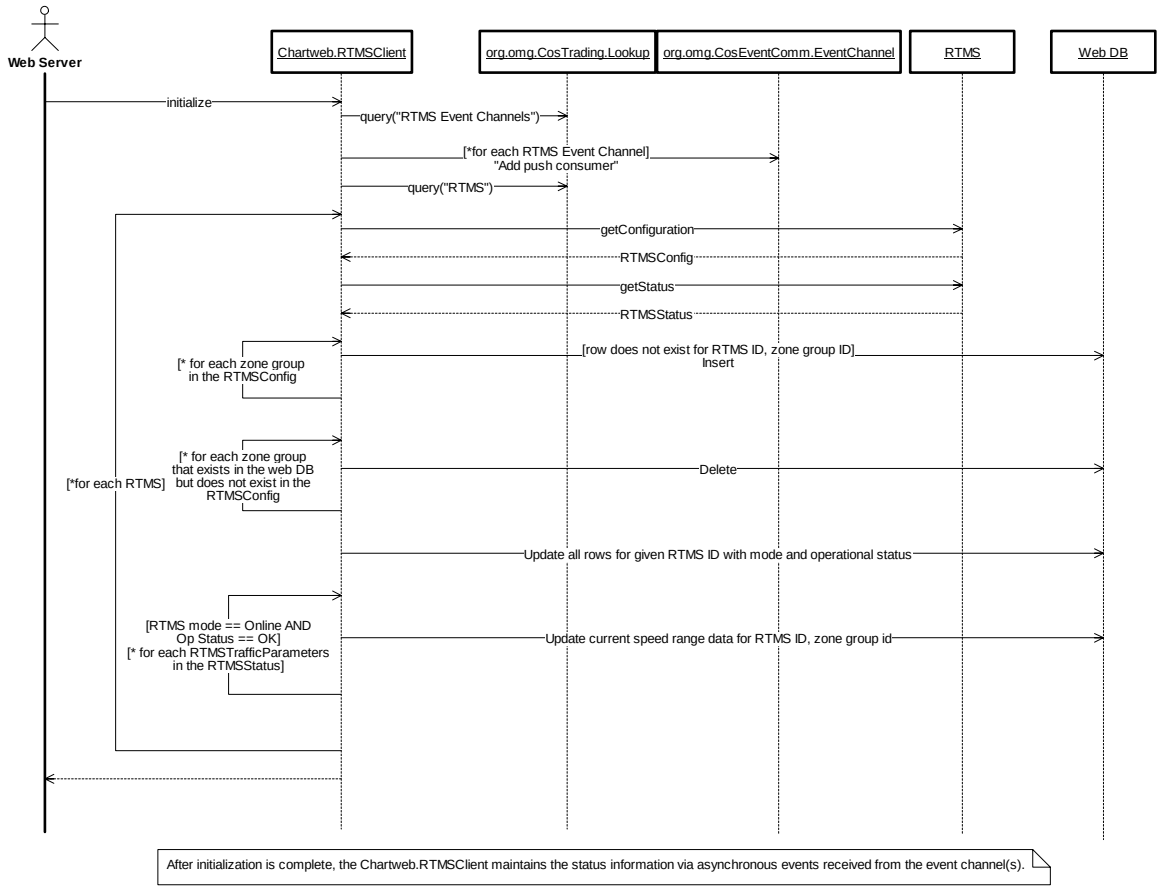

Figure 57. CHARTWebModule:Initialize (Sequence Diagram)

3.5.7 ChartWeb Database Tables

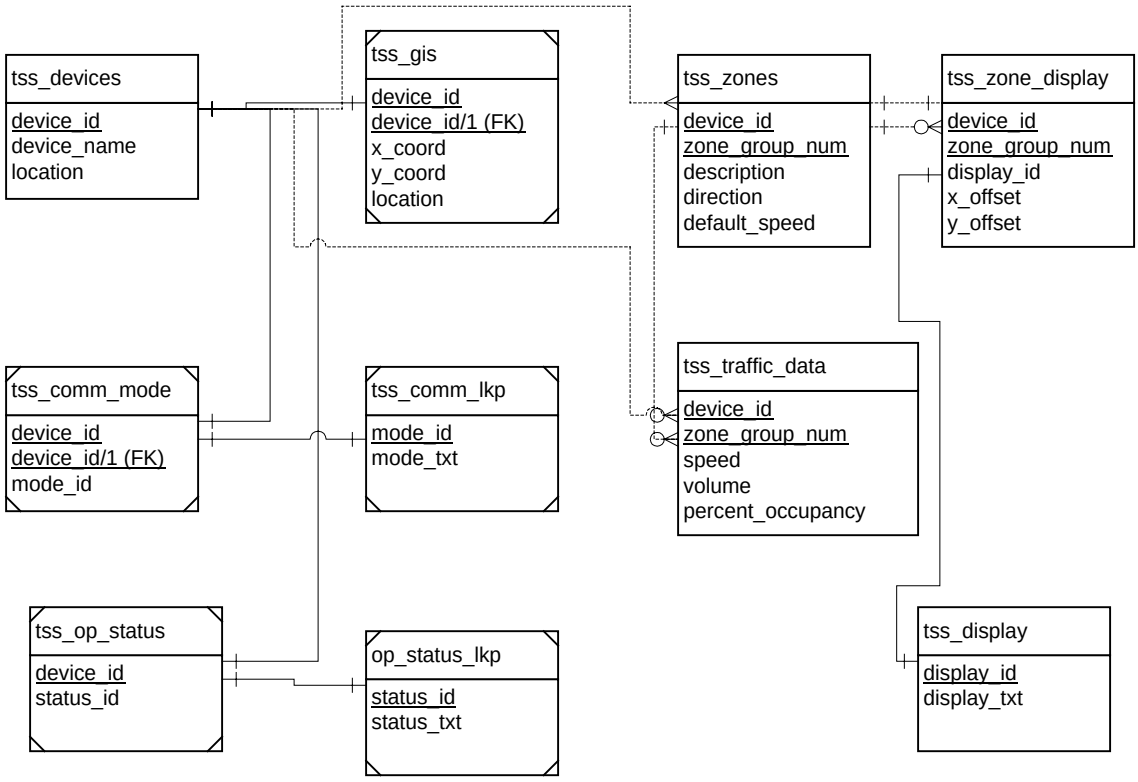

Figure 58. CHARTWeb Database Tables

3.6 CHART Web Map Server

3.6.1 CHART Web Map Server (Class Diagram)

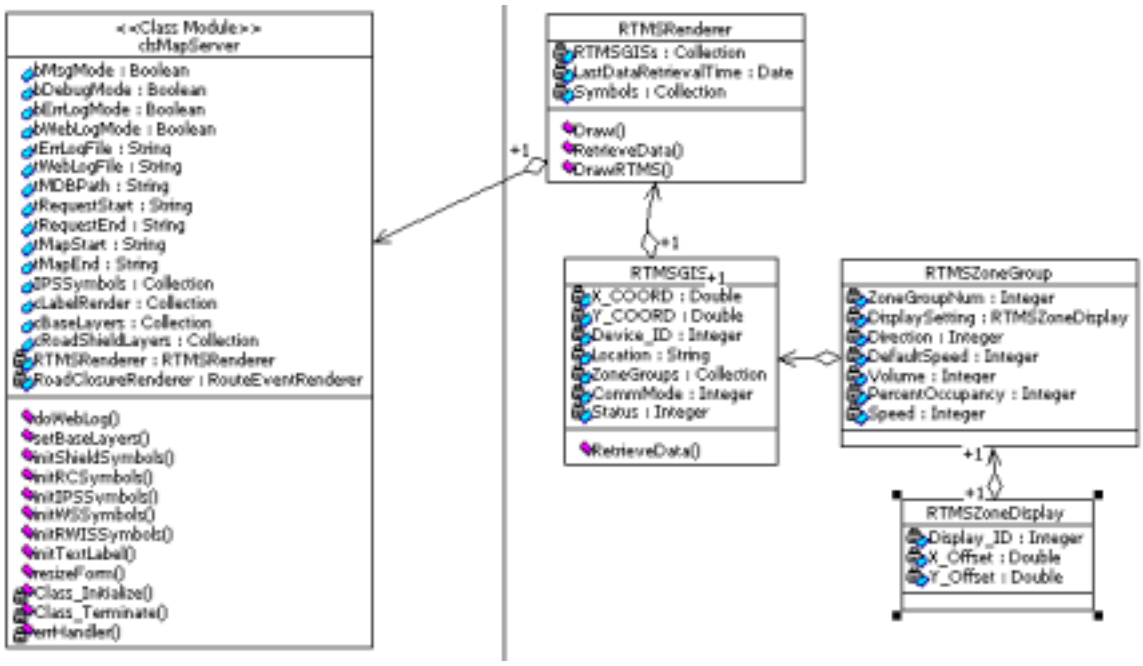

Figure 59. CHART Web Map Server Business Classes

3.6.1.1 Business Service Classes

3.6.1.1.1 clsMapServer (Class)

This is the top class for the map server application. Contains all components in the web mapping application

3.6.1.1.2 clsWebRequest (Class)

This class holds the parameters of the web map request, such as the boundary of the map, layer on/off settings, etc.

3.6.1.1.3 clsWebResponse (Class)

This class holds the parameters of the web map response, such as the return parameters, map image location, etc.

3.6.2 CHART II Web Map Server Data Service Classes

3.6.2.1 RTMSZoneDisplay

This class stores display configurations for each zone group, including DisplayID that enumerates the display settings for zone groups with valid values NONE, INTERNAL, INTRANET, and INTERNET. Also included in this class is the X/Y offset for each group to adjust the map display so that clustered symbols could be avoided or reduced.

3.6.2.2 RTMSZoneGroup

This class holds data for RTMS zone groups, such as speed, volume, and percent occupancy.

3.6.2.3 RTMSGIS

This class holds the GIS data related with each RTMS device, including its X/Y coordinates, device ID, location description and RTMSZoneGroups for this RTMS device.

3.6.3 CHART II Web Map Server User Service Classes

3.6.3.1 RTMSRenderer

This class is used to paint the RTMS symbols to the map for display. It also holds a cached version of the RTMSGIS data collection.

3.6.4 CHARTWebMapServerModule:ServiceInternetRequest (Sequence Diagram)

When web map server receives web map request, MapServer calls the Draw method of RTMSRenderer. RTMSRenderer check its cached RTMSGISData, if RTMSRenderer data is too old, it will call RetrieveData method of RTMSGIS to retrieve a new set of RTMSGIS data including location information for the device, zone groups and their status data. RTMSRenderer loops through the RTMSGIS collection, based on default speed, speed, volume, percent occupancy information, call DrawRTMS method to display the zone group with appropriate symbol on the map

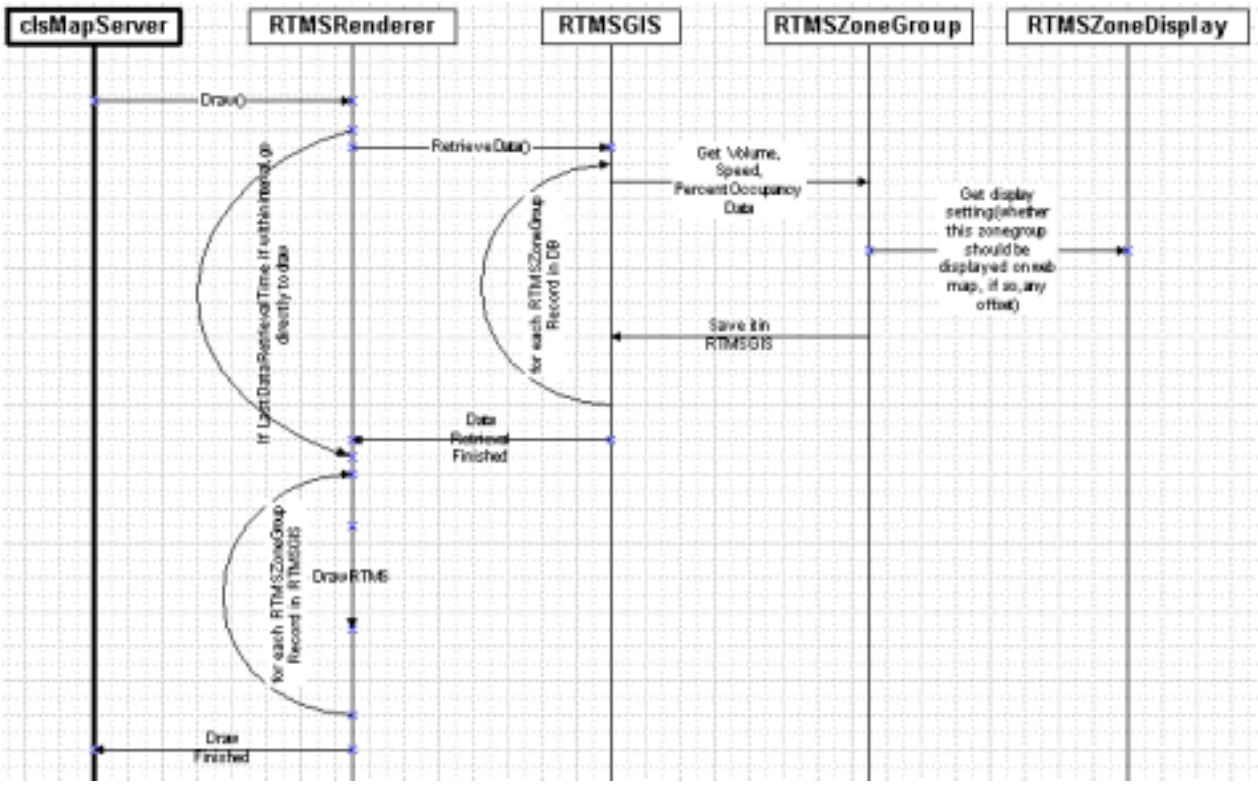

Figure 60. CHARTWebMapServer:ServiceInternetRequest (Sequence Diagram)

Acronyms

The following acronyms appear throughout this document:

BAA	Business Area Architecture
CORBA	Common Object Request Broker Architecture
DBMS	Database Management System
FMS	Field Management Station
GUI	Graphical User Interface
IDL	Interface Definition Language
ITS	Intelligent Transportation Systems
ORB	Object Request Broker
POA	Portable Object Adapter
RTMS	Remote Traffic Microwave Sensor
R1B2	Release 1, Build 2 of the CHART II System
R1B2A	Release 1, Build 2A of the CHART II System
TSS	Transportation Sensor System
UML	Unified Modeling Language

References

CHART II Business Area Architecture Report, document number M361-BA-005R0, Computer Sciences Corporation and PB Farradyne.

CHART II System Requirements Specification Release 1 Build 2, document number M361-RS-002R1, Computer Sciences Corporation and PB Farradyne.

R1B2 High Level Design, document number M362-DS-005R0, Computer Sciences Corporation and PB Farradyne.

R1B2A High Level Design, document number M303-DS-004R0, Computer Sciences Corporation and PB Farradyne.

R1B2 Servers Detailed Design, document number M362-DS-006R0, Computer Sciences Corporation and PB Farradyne.

R1B2 GUI Detailed Design, document number M362-DS-007R0, Computer Sciences Corporation and PB Farradyne.

FMS R1B2 High Level Design, document number M303-DS-002R0, Computer Sciences Corporation and PB Farradyne.

The Common Object Request Broker: Architecture and Specification, Revision 2.3.1, OMG Document 99-10-07

Martin Fowler and Kendall Scott, *UML Distilled,* Addison-Wesley, 1997

Appendix A – Functional Rights

This table lists the functional rights that have been added to the CHART II system for R1B2A (or existed previously and apply to R1B2A) and the operations to which they grant access.

Functional Right Required	Operation	Organization Filterable
ConfigureTSS (new to R1B2A)	Add TSS	No
	Remove TSS	No
	Set TSS Configuration	No
	Get TSS Configuration	No
ViewTSSConfig (new to R1B2A)	Get TSS Configuration	No
Maintain TSS (new to R1B2A)	Put a TSS in Maintenance Mode	No
	Move a TSS from maintenance mode to online.	No
	Move a TSS from maintenance mode to offline.	No
ManageDeviceComms (exists in R1B2)	Put a device Online	No
	Take a device Offline	No

Appendix B – Glossary

CORBA Event	A CORBA mechanism using which different Chart2 components exchange information without explicitly knowing about each other.
CORBA Trader	A CORBA service that facilitates object location and discovery. A server advertises an object in the Trading Service based on the kind of service provided by the object. A client locates objects of interest by asking the Trading Service to find all objects that provide a particular service.
Data Model	An object repository that keeps track of changes to the various objects in the repository and informs about these changes as they occur, to observers who are interested in the objects in the repository. A Data Model identifies the subject in a Subject/Observer design pattern.
Factory	A CORBA object that is capable of creating other CORBA objects of a particular type. The newly created object will be served from the same process as the factory object that creates it.
FMS	Field Management Station through which the CHART II system communicates with the devices in the field.
Functional Right	A privilege that gives a user the right to perform a particular system action or related group of actions. A functional right may be limited to pertain only to those shared resources owned by a particular organization or can pertain to the shared resources of all organizations.
Graphical User Interface	Part of a software application that provides a graphical interface to its user.
GUI Wrapper Object	A GUI wrapper object is one that wraps a server object to provide it with GUI functionality such as menu handling. It also helps in performance enhancement by caching data locally thereby avoiding network calls when not necessary.
Installable Module	A plugable GUI module that provides a specific function, which when registered with the GUI is called on to initialize itself at the time of GUI startup and shut down at the time of GUI shut down.

Navigator	A Navigator is a GUI window that contains a tree on the left-hand side and a list on the right hand side. Tree elements represent groups of objects and the list on the right hand side represents the objects in the selected group.
Object Discovery	A GUI mechanism in which the client periodically asks the CORBA Trading Service to find objects of those types that are of interest to the GUI, such as DMS, HAR, Plan etc.
Operator	A CHART II user that works at an Operations Center.
Port	A software object used to model a physical communications port.
Port Manager	A software object that manages access to one or more communications ports.
Protocol Handler	A software object that contains code that encapsulates the specific communications sequences required to command a field device.
RTMS	A traffic sensor capable of sensing volume, speed, and occupancy for up to 8 lanes of traffic.
Service Application	A software application that can be configured to run one or more service application modules and provides them basic services needed to serve CORBA objects.
Service Application Module	A software module that serves a related group of CORBA objects and can be run within the context of a service application.
Token	A token or access token is a security blob that encloses information about a user and the functional rights associated with the user. All secured Chart2 operations require a token to be passed to it and based on the functional rights found in a token a user is allowed or denied access.
Transportation Sensor System	A system capable of sensing and communicating traffic parameters.
User	A user is someone who uses the CHART II system. A user can perform different operations in the system depending upon the roles they have been granted.